UNIVERSITY OF CALGARY

Dependency Detection and Migration in Software Systems and Libraries

by

Bradley E. Cossette

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

September, 2014

© Bradley E. Cossette 2014

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Dependency Detection and Migration" submitted by Bradley E. Cossette in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Dr. Robert James Walker
Supervisor
Department of Computer Science

Dr. Günther Ruhe
Examination Committee Member
Department of Computer Science

Dr. Philipp Woelfel
Supervisory Committee Member
Department of Computer Science

Dr. Diwakar Krishnamurthy
Internal / External Examiner
Dept. of Electrical Engineering

Dr. Jörg Denzinger
Supervisory Committee Member
Department of Computer Science

Dr. Daniel Dig
External Examiner
Oregon State University

Date

Abstract

Software systems must change over time or risk becoming obsolete, but direct changes can impact dependent functionality. Software developers perform *change impact analysis and redress* using automated tools to identify dependency relationships affected by change, and to recommend adaptations. However, these tools are restricted in their application: dependency analysis tools are language-specific but many systems are implemented using multiple languages, while recommenders are poor at identifying adaptations for change impacts from external libraries.

This work proposes using semi-automated approaches for supporting change impact analysis and redress, in which the developer is relied upon to provide key details of dependency syntax or examples of correct adaptations. From such information, tool support can be generated that is appropriately configured for the developer's software, and that can be further refined by the developer through additional details or examples until it is sufficiently accurate for their needs.

Four studies validate this thesis. The first involves the semi-automated DSKETCH tool for polylingual dependency analysis, which requires developers to detail only key syntax using a simplified notation; it then uses this syntax to identify where potential dependencies exist. Developers were able to successfully configure and use DSKETCH on polylingual systems with only a short period of training. The second study examines how software libraries and their application programming interfaces (APIs) evolve over successive versions. The study found that existing recommenders are generally unsuccessful, and that most observed changes could not be automatically migrated. The third study introduces the UMAMI tool that detects correspondences between the syntactic structure of the old API functionality and possible replacements in the new library version to recommend adaptations for API changes. The fourth study examines how change recommenders could be hybridized in a flexible fashion, by relying on developer-provided examples of correct redress of API changes to tailor recommendations to a library's particular characteristics.

Acknowledgments

> We cannot tell the precise moment when friendship is formed. As in filling a vessel drop by drop, there is at last a drop which makes it run over; so in a series of kindnesses there is at last one which makes the heart run over.
>
> James Boswell, *The Life of Samuel Johnson*

Completing this dissertation has been the story of many kindnesses from friends, family, and God, that helped me on the way. First, I am grateful to my supervisor Robert Walker, for his mentorship, support, and especially his friendship over the last eight years. I could write much about his character, and integrity that I now try to emulate in my own work, but I appreciate most his years of encouragement, and his constant goal of training me to be his colleague, and not just his student: I see echoes of his writing and perspective in my work every day.

Many lab-mates have come and gone over the years. Rylan Cottrell, who started his Master's and PhD degrees with me, and without whom I would have never survived Cryptography, is the first and foremost I must thank. For eight years you were my friend and co-conspirator, and I will miss your boundless willingness to talk about anything: whether God and religion, relationships, research, or the importance of Dr. Who to science fiction. Reid Holmes still acts as my big brother, introducing me to the research community at large, and always willing to offer advice. Puneet Kapur, who is the first programmer and computer scientist I know to also be a medical doctor, was a font of friendship, humour, insight, and wisdom. Thanking the cast of characters in our lab would not be complete without mentioning Craig Schock, Soha Makady, Hamid Baghi, Elham Moazzen, and Mostafa Sayed. Thank you as well to Leanne Wu, Tyson Kendon, Sean Nichols, and everyone else who also shared the joys and travails of grad life with me at Tuesday Tea.

I could not have prouder parents in Bernard and Diana Cossette, and an even prouder grandmother in Grace Kine, who have supported me financially, emotionally, and through prayer. My grandparents and great-grandparents have loved books, knowledge, and teaching, and it was a special moment when I realized that my father was also a published author of technical papers in his

field; I follow in all their footsteps. My brothers and their wives—Brandon and Christine Cossette, Aaron and Alicia Cossette—have been a comfort and encouragement without fail, and your joy over my accomplishments is a shadow of how proud I am of you and your children (Kael, Courtney, and Isaiah). I am also grateful for my wife's family: Kent and Susan Dunfield, Michael, Susan, and Jacob Dunfield, and Robert Dunfield, who treated me as one of their own from the day we met, even trying to read my research papers to show their support.

Denis and Laura Perron, thank you for your prayers, hospitality, and most of all for introducing me to my wife. Justin and Kellie Agnew Clark, thank you for being so accepting and gracious: your company is always treasured. I am especially grateful for the family of friends and mentors I had at Bethany Chapel in Calgary, who showered me with kindness and grace throughout my studies.

But I have saved my deepest thanks for my greatest supporter, my wife Charlotte who fell in love with this PhD student. You chose to walk with me the whole way on this adventure, following me to Calgary from Vancouver, sharing your amazing circle of friends with me, embracing my family, and now following me again to Montréal and McGill University. I love you, and you are the only one who knows every joy and sorrow I faced during this degree, because you chose to share for better and worse with me. Sharing this with you is the greatest joy of this degree.

Finally, I want to thank my God, and my Saviour Jesus Christ, who has been faithful to carry me through this entire journey, even when I doubted I would ever finish. It was a message from Jeremiah 32:1–15 six years ago that convinced me that God would see me through my PhD; may I always remember His faithfulness to bring me through the impossibilities in life.

BRADLEY COSSETTE

September 22, 2014

Montréal, Québec

For Charlotte, who took my hand,

and walked the way with me.

Table of Contents

Abstract . . . ii
Acknowledgments . . . iii
Table of Contents . . . vi
List of Tables . . . x
List of Figures . . . xii
1 Introduction . . . 1
1.1 Overview of Related Work . . . 3
1.2 Semi-Automating Developer Configured Solutions . . . 6
1.3 Thesis Statement . . . 8
1.4 Organization . . . 9
1.5 Publications . . . 9
2 Motivation . . . 11
2.1 Proactive Change: Polylingual Dependency Analysis . . . 11
2.2 Reactive Change: API Migration and Redress . . . 14
2.3 Summary: Motivation . . . 18
3 Related Work . . . 20
3.1 Change Impact and Dependency Analysis . . . 21
3.1.1 Formulating Change Impact . . . 22
3.1.2 Reverse Engineering and Tools For Dependency Analysis . . . 24
3.1.3 Summary: Change Impact and Dependency Analysis . . . 30
3.2 Polylingual Dependency Analysis . . . 31
3.2.1 Lexically-Aware Tools in Polylingual Contexts . . . 33
3.2.2 Syntactically-Aware Tools in Polylingual Contexts . . . 34
3.2.3 Semantically-Aware Tools in Polylingual Contexts . . . 37
3.2.4 Partial-Language Recognition Approaches . . . 39
3.2.5 Summary: Polylingual Dependency Analysis . . . 42
3.3 Library Evolution and API Migration . . . 43
3.3.1 Proactive Library Migration . . . 45
3.3.2 Reactive Library Migration . . . 47
3.3.3 Recommending appropriate reactions . . . 50
3.3.4 Redressing library change impact . . . 55
3.3.5 Summary: Library Migration and API Evolution . . . 57
3.4 Empirical Studies of API Change . . . 58
3.4.1 Empirically Determining How Libraries Have Changed . . . 59
3.4.2 Evaluating in the Absence of *A Priori* Knowledge . . . 62
3.4.3 Summary: Empirical Studies of API Change . . . 64
3.5 Summary: Related Work . . . 65
4 Supporting Proactive Polylingual Change Impact Analysis . . . 67
4.1 The Sketch Notation . . . 69
4.1.1 Sketch Notation Example . . . 71
4.2 Detecting Identifiers . . . 72
4.2.1 Compiling Sketched Patterns . . . 73

4.2.2 Pattern Collisions . . . 74
4.3 Detecting Dependencies . . . 75
4.4 Using DSKETCH . . . 76
4.5 Evaluation . . . 79
4.5.1 Case Study Overview . . . 80
4.5.2 Case Study 1 . . . 81
4.5.3 Case Study 2 . . . 85
4.6 Analysis of Observations . . . 87
4.6.1 RQ 4.1: Can developers configure the tool? . . . 88
4.6.2 RQ 4.2: How much effort is required to adapt? . . . 88
4.6.3 RQ 4.3: Does the tool produce results that are reasonably correct? . . . 88
4.6.4 RQ 4.4: Does the approach work? . . . 89
4.7 Discussion . . . 89
4.7.1 Qualitative vs. Quantitative Evaluation . . . 89
4.7.2 Improving pattern expressiveness . . . 90
4.7.3 Usability and performance . . . 90
4.7.4 Adapting to large-scale system issues . . . 91
4.8 Summary: Proactive Change . . . 91
5 The Evolution and Migration of Software Libraries . . . 93
5.1 Methodology . . . 95
5.1.1 Determining Points of Change . . . 96
5.1.2 Finding Replacement Functionality . . . 97
5.1.3 Classifying Transformations . . . 100
5.1.4 Verification of Recommendations . . . 101
5.2 Observations and Results . . . 103
5.2.1 Overview of Changes . . . 103
5.3 Recommendation Techniques . . . 104
5.3.1 Effectiveness of Call Analysis . . . 105
5.3.2 Effectiveness of Code Comments . . . 107
5.3.3 Effectiveness of Release Documentation . . . 107
5.3.4 Effectiveness of Lexical Search (Via the Entity's Name) . . . 108
5.3.5 Effectiveness of Lexical Search (Via the Code Comment) . . . 108
5.3.6 Effectiveness of Implementation Details . . . 109
5.3.7 Effectiveness of Experience . . . 109
5.3.8 RQ 5.1: How effective are recommendation techniques? . . . 109
5.4 API Transformations . . . 110
5.4.1 Fully Automatable . . . 110
5.4.2 Partially Automatable . . . 111
5.4.3 Hard to Automate . . . 113
5.4.4 RQ 5.2: How feasible is it to automatically migrate dependencies? . . . 116
5.5 Discussion . . . 117
5.5.1 Recommendation Blindness . . . 118
5.5.2 Discrepancies with the Literature . . . 119
5.5.3 Threats to Validity . . . 124
5.6 Summary: Empirical Study of Reactive Change . . . 126

6 Using Structural Analysis for Reactive Change . . . 128
6.1 Background: Anti-Unification . . . 129
6.1.1 Applications of anti-unification . . . 131
6.2 Matching via Structural Generalization . . . 132
6.2.1 Computing Similarity between Nodes . . . 133
6.2.2 Determining Correspondence . . . 135
6.3 Is MSG Appropriate for API Change Recommendation? . . . 137
6.3.1 Observations . . . 139
6.3.2 RQ 6.1, 6.3: Can MSG resolve changes comparatively well? . . . 141
6.3.3 RQ 6.2: Does MSG's assumptions impact its effectiveness? . . . 142
6.4 UMAMI: API-Aware Structural Matching . . . 144
6.4.1 Engineering MSG for APIs . . . 145
6.4.2 The UMAMI algorithm . . . 146
6.5 Evaluation of Umami . . . 148
6.5.1 Performance . . . 150
6.5.2 RQ 6.4, 6.5: Can UMAMI resolve changes comparatively well? . . . 150
6.5.3 RQ 6.6: How does UMAMI compare to other API change recommenders? . . . 152
6.6 Discussion . . . 155
6.6.1 Threats to Validity . . . 155
6.6.2 Comparison with Cottrell et al. [2008] . . . 156
6.6.3 Future Work . . . 157
6.7 Summary: Using Structural Analysis for Reactive Change . . . 158
7 Hybridizing Recommenders for Reactive Change . . . 161
7.1 Evaluating Inference Techniques . . . 162
7.1.1 Baseline Implementations . . . 163
7.1.2 Observations . . . 167
7.1.3 RQ 7.1: How effective are inference techniques in resolving API change? . . . 167
7.2 Flexible Hybridization . . . 169
7.2.1 Challenges in Hybridization . . . 170
7.2.2 Machine Learning for Change Recommenders . . . 171
7.3 Case Study . . . 172
7.3.1 Methodology . . . 172
7.3.2 Results . . . 173
7.3.3 RQ 7.2: Could end-users successfully configure hybrid combinations? . . . 177
7.4 Discussion . . . 177
7.5 Summary: Hybridizing Recommenders for Reactive Change . . . 179
8 Discussion . . . 180
8.1 The Library Migration Corpus . . . 181
8.2 Future Work . . . 182
8.2.1 Case Studies of Semi-Automated Redress for API Migration . . . 182
8.2.2 Technical Risk Estimation in API Migration . . . 184
8.3 Summary: Discussion . . . 186
9 Conclusions . . . 188
9.1 Contributions . . . 191
A Overview of Appendices . . . 210

B Certification of Ethics Approval . . . 211
C List of Publications and Copyright Permissions . . . 212
D Comparing Existing Work to the Library Migration Corpus . . . 213
D.1 Overview . . . 213
D.1.1 On Notation . . . 215
D.2 Comparison with Dig et al. . . . 216
D.2.1 Changes on which the Corpus and Dig et al. Agree . . . 216
D.2.2 Non-Library Changes Considered by Dig et al. . . . 217
D.2.3 Non-BI Changes Considered by Dig et al. . . . 218
D.2.4 Comparison with Dig et al.: Summary . . . 226
D.3 Comparison with Schäfer et al. . . . 227
D.3.1 Changes on which Schäfer et al. and the Corpus Agree . . . 228
D.3.2 Non-BI Changes Considered by Schäfer et al. . . . 230
D.3.3 Changes on which the Corpus and Schäfer et al. Disagree . . . 236
D.3.4 Comparison with Schäfer et al. [2008]: Summary . . . 236
D.4 Comparison with Wu et al. . . . 237
D.4.1 Changes on which Wu et al. and the Corpus Agree . . . 238
D.4.2 Non-Library Changes Considered by Wu et al. . . . 241
D.4.3 Non-API Changes Considered by Wu et al. . . . 243
D.4.4 Changes on which the Corpus and Wu et al. Disagree . . . 250
D.4.5 Comparison with Wu et al.: Summary . . . 265
D.5 Comparison with Meng et al. . . . 266
D.5.1 Changes on which Meng et al. and the Corpus Agree . . . 267
D.5.2 Non-Library Changes Considered by Meng et al. . . . 271
D.5.3 Non-API Changes Considered by Meng et al. . . . 280
D.5.4 Changes on which the Corpus and Meng et al. Disagree . . . 282
D.5.5 Comparison with Meng et al.: Summary . . . 283
D.6 Summary: Comparing Existing Work to the Library Migration Corpus . . . 284
E The Library Migration Corpus . . . 290
E.1 Notation and Formatting . . . 290
E.2 Table Column Explanations . . . 292
E.3 The Library Migration Corpus . . . 293
E.3.1 APACHE LUCENE . . . 293
E.3.2 APACHE STRUTS . . . 293
E.3.3 DBCP . . . 294
E.3.4 JDOM . . . 294
E.3.5 LOG4J . . . 294
E.3.6 SLF4J . . . 294

List of Tables

5.1 Summary of BIs. + = public entities; # = protected entities. . . . 103
5.2 BIs resolved by a given number of techniques. . . . 104
5.3 Effectiveness of techniques to detect resolutions. F = found; M = missing; E = error; CC = code comment; Ex = experience. . . . 106
5.4 Transformation types. . . . 117
5.5 Automatic transformability of BIs. F = fully automatable; P = partially automatable; H = hard to automate. . . . 118

6.1 Comparison of MSG against other inference techniques. . . . 140
6.2 Categories of change characteristics that MSG fails to resolve. . . . 142
6.3 Comparative results. U = UMAMI; BI = binary incompatibilities on methods; CA = call analysis; EC = explicit in comments; TSC = text search on comments; TSN = text search on entity name; RD = release documents. . . . 159
6.4 Comparison to AURA, Schäfer et al., and REFACTORINGCRAWLER (RC) for STRUTS 1.1.0 to 1.2.4. . . . 160

7.1 Prediction results from the implemented inference techniques. Σ BIs = total binary incompatibilities, CA = call analysis, SS = structural similarity, LED = lexical edit distance, LCS = longest common substring, Sig = signature, Com = comments, DC = deprecation comments, T = true recommendations, F = false recommendations. . 168
7.2 Comparison of ML technique accuracy, as percentages. CA = call analysis, SS = structural similarity, Sig = LED and LCS analysis of method signatures, Com = LED and LCS comparison of comment text for affected methods. . . . 174
7.3 Comparison of ML technique accuracy across all APIs, as percentages. . . . 175
7.4 Comparison to REFACTORINGCRAWLER, Schäfer et al., and AURA for 1:1 method-level BI changes in STRUTS 1.1.0 to 1.2.4. . . . 176

D.1 Dig et al. Matched API Changes. (All in org.apache.struts) . . . 216
D.2 Dig et al. Non-Library Changes. (All in org.apache.struts.webapp.tiles) . . . 218
D.3 Dig et al. Non-BI Changes. (All in org.apache.struts) . . . 219
D.4 Comparison of REFACTORINGCRAWLER to the Corpus for STRUTS 1.1.0 to 1.2.4. 226
D.5 Schäfer et al. API Changes in Agreement with the Corpus. (All in org.apache.struts) 228
D.6 Non-BI Changes Recorded by Schäfer et al.. (All in org.apache.struts) . . . 231
D.7 Comparison of Schäfer et al. to the Corpus for STRUTS 1.1.0 to 1.2.4. . . . 236
D.8 Wu et al. Matched API Changes. (All in org.apache.struts) . . . 238
D.9 Wu et al. Changes External to the STRUTS API. (All in org.apache.struts) . . . 241
D.10 Wu et al. Changes to the STRUTS test suite. (All in org.apache.struts) . . . 242
D.11 Wu et al. Non-API Changes. (All in org.apache.struts) . . . 244
D.12 Wu et al. API Changes in Disagreement. (All in org.apache.struts) . . . 251
D.13 Comparison of AURA to the Corpus for STRUTS 1.1.0 to 1.2.4. . . . 265
D.14 Meng et al. Matched API Changes. (All in org.apache.struts) . . . 267
D.15 Meng et al. Deleted API methods for which replacement functionality is external to STRUTS. . . . 271

D.16 Meng et al. Matched API Changes. (All in org.apache.struts) 271
D.17 Meng et al. Deleted methods from STRUTS examples. (All in org.apache.struts-.webapp.example) 279
D.18 Meng et al. Deleted methods from the STRUTS test suite (All in org.apache.struts-.taglib). 280
D.19 Meng et al. Matched API Changes. (All in org.apache.struts) 281
D.20 Meng et al. Deleted methods with **private** visibility in the STRUTS library (All in org.apache.struts). 282
D.21 Meng et al. Unchanged method reported as changed (All in org.apache.struts). . . . 283
D.22 Meng et al. Methods marked deleted, that were instead refactored into their super-class in the STRUTS API (All in org.apache.struts). 283
D.23 Meng et al. Methods marked deleted, for which replacements existed in the STRUTS API (All in org.apache.struts). 288
D.24 Comparison of HIMA to the Corpus for STRUTS 1.1.0 to 1.2.4. 289
D.25 Comparison of all techniques against the Corpus for STRUTS 1.1.0 to 1.2.4. 289

E.1 APACHE LUCENE 3.1.0 to 3.2.0 migration. 295
E.2 APACHE LUCENE 3.2.0 to 3.3.0 migration. 301
E.3 APACHE LUCENE 3.3.0 to 3.4.0 migration. 312
E.4 APACHE STRUTS 1.0.2 to 1.1.0 migration. 315
E.5 APACHE STRUTS 1.1.0 to 1.2.4 migration. 391
E.6 APACHE STRUTS 1.2.4 to 1.2.6 migration. 460
E.7 APACHE STRUTS 1.2.6 to 1.2.7 migration. 463
E.8 JDOM b6 to b7 migration. 472
E.9 JDOM b7 to b8 migration. 477
E.10 JDOM b8 to b9 migration. 485
E.11 LOG4J 1.0.4 to 1.1.3 migration. 490
E.12 LOG4J 1.1.3 to 1.2.1 migration. 498
E.13 LOG4J 1.2.1 to 1.2.2 migration. 521
E.14 LOG4J 1.2.4 to 1.2.5 migration. 522
E.15 LOG4J 1.2.5 to 1.2.6 migration. 523
E.16 LOG4J 1.2.8 to 1.2.9 migration. 524

List of Figures and Illustrations

2.1 Comparing ActionServlet.processLocale() to RequestProcessor.processLocale() . . 18

4.1 Screenshot of DSKETCH. 76
4.2 Sketch list. 77
4.3 Language mappings. 77
4.4 Creating or Editing a Sketch pattern. 77
4.5 Dependency highlighting in source code. 78
4.6 Detected dependencies. 79
4.7 All dependencies. 79

6.1 Example ASTs (AST_1 and AST_2) and a possible anti-unifier (AST_{AU}). 129

Chapter 1

Introduction

> We are confronting a universe marked by tremendous fluidity ... where fragmentation, splintering, and disappearance are the mirror images of appearance, emergence, and coalescence.
>
> Anselm L. Strauss [1993, p. 19]

Software systems must change over time if they are to remain useful and relevant to their users [Belady and Lehman, 1976]. This need for change does not mean that the original release of the software system failed to solve the problem, or fulfill the purpose, for which it was designed; rather, it is the success of these systems that creates pressure from its users for improvements beyond what was initially provided [Parnas, 1994]. The external pressure for change is not unreasonable: as defects in the original software are discovered, users will wish to see them fixed; and users will also expect additional features to be added to the software to keep up with competing offerings [Lehman, 1980, 1996]. Modifying the software's implementation is not easy though; often, developers find that their modifications have unintentional consequences that propagate throughout the system.

Software systems are normally designed as a composition of smaller pieces of functionality, or *software entities*, that are combined to solve the tasks the software needs to perform [Parnas, 1972; Parnas and Siewiorek, 1975]. Creating systems out of entities also gives the developer freedom to reuse existing entities: they may have created similar functionality before [Holmes and Walker, 2007], or they can leverage collections of entities in *software libraries* that provide needed functionality [Krueger, 1992]. However, this also results in entities being *dependent* on other entities for their correct functioning; some entities will unavoidably have many such dependencies [Parnas, 1979; Brooks, 1987; Taube-Schock et al., 2011]. The risk is that any change to one entity's functionality can have an unintended impact on every other entity that was dependent on

it (called the *ripple effect*); thus, changes that do not take into account this network of dependency relationships can introduce unanticipated side-effects, often resulting in adding new defects to the software, each of which must be addressed [Podgurski and Clarke, 1990; Arnold and Bohner, 1993; Parnas, 1994; Robillard and Murphy, 2007].

Preventing such side effects requires that the developers identify the impact of changes made to the software, and determine how to redress them [Arnold and Bohner, 1993; Chow and Notkin, 1996]. But how developers go about *change impact analysis and redress*, and the difficulties they face, depends in large part on whether the change can be anticipated by the developer. Ideally, software developers *proactively* change their software systems by planning their change in advance, identifying the dependency relationships that may be affected, and determining how to mitigate their change's impact before even starting. However, these investigations are only approximations of the actual dependencies present, and are difficult to perform automatically or manually [Brooks, 1987; Luqi, 1990; Moriconi and Winkler, 1990; Holmes and Walker, 2007].

Some changes cannot be anticipated: when serious bugs are identified, or when relied upon software libraries change in incompatible ways, software developers have limited choices, or sometimes none at all, in determining how the change needs to be enacted [van Gurp and Bosch, 2001; Xing and Stroulia, 2007b; Fry and Weimer, 2010; Guo et al., 2010; Park et al., 2012; Murphy-Hill et al., 2013]. In these cases, developers must *react* to the change, rather than proactively plan it; they must identify how to redress the impact of a change they cannot mitigate through other means. This is particularly problematic when the change is introduced from external systems, such as software libraries, as there is often no indication as to what has changed, nor how it should be redressed [Andersen and Lawall, 2010; Kapur et al., 2010; Dagenais and Robillard, 2011].

To assist developers in change impact analysis and redress tasks, researchers have provided different tools that can either identify where dependencies exist in their software [Chikofsky and Cross, 1990] or recommend adaptive changes to redress the change impact. Existing work has focused primarily on providing *fully automated* tools for these cases, which are preferred by develop-

ers due to their ease of use [Chen et al., 1990; Chikofsky and Cross, 1990]. However, constructing general purpose tool support is very expensive due to the sheer variety of languages, technologies, and kinds of software changes that must be supported: implementing the detailed semantic knowledge necessary for this analysis is combinatorially expensive [Biggerstaff, 1994; Cossette and Walker, 2007]. Some alternative proposals suggest using flexible, widely applicable tools that expect developers to specify precise patterns for use in analysis: however, the pattern syntax is often difficult to use, and easily confused by superficial similarities [Cossette and Walker, 2007].

In this dissertation, I present *semi-automated* tool support for change impact and redress tasks: I show how developers can be provided with tools that they can configure through either feedback-based training or a simplified specification notation. Developers can apply these tools to identify approximate solutions in their change impact and redress tasks, iteratively refining their configuration to improve the tool's analyses until it is sufficiently effective for their needs. Semi-automated approaches for tool support have been successfully applied to the problems of helping developers proactively investigate change impact in software implemented with multiple programming languages, and in reactively identifying adaptive changes for software library migrations through a feedback-based hybridization of recommender systems.

1.1 Overview of Related Work

Research on the problem of change impact initially focused on proactive prevention by anticipating future changes in the software's design [Parnas, 1972; Stevens et al., 1974; Parnas, 1979], but later recognized that this is not always possible, requiring the development of techniques to detect dependency relationships to identify the impact of proposed changes [Brooks, 1987; Arnold and Bohner, 1993; Parnas, 1994]. Early work in dependency analysis took a highly formal approach, but was quick to recognize the need for approximation, and the important effects that subtly differing definitions of "dependency" could have [Podgurski and Clarke, 1990; Moriconi and Winkler, 1990; Landi, 1992]. More pragmatic approaches to impact analysis followed [Arnold and Bohner,

1993], but still made strong assumptions about having the intended change implemented in order to perform the analysis.

Initial tool-based approaches for proactive investigation of dependencies focused on cross-referencing declarations to their usages by reverse-engineering models of source code [Chikofsky and Cross, 1990; Chen et al., 1990], but full semantic analysis of a language proved too expensive to construct, and adapting existing tools for new languages or dialects was also expensive [Grass and Chen, 1990; Reubenstein et al., 1993]. Later work recognized that lexically or syntactically-aware approaches to dependency analysis were significantly cheaper to implement, and often sufficient for extracting key dependency details from source code [Wong et al., 1995].

Polylingual software systems however pose a difficult problem. The language and technology combinations possible make general-purpose semantic support impractical to implement [Biggerstaff et al., 1994; Cossette and Walker, 2007], relegating it to specific sub-problems [Furr and Foster, 2006], while lexical tools require onerous specifications to reduce inaccuracies [Murphy and Notkin, 1996], and syntactic tools can still require significant rewrites to accommodate additional languages [Atkinson and Griswold, 2006; Moise and Wong, 2005]. Promising approaches based on partial-language specification techniques have shown to be effective in polylingual contexts [Koppler, 1997; Moonen, 2002], but the costs of improving the accuracy of specifications for dependency analysis are still high [Cossette and Walker, 2007]. A practical application requires simplifying how polylingual dependency characteristics must be specified.

Research on reacting to the impact of changes is prevalent in the study of software library migrations. Here, the reusable functionality provided through a library's application programming interface (API) can change between versions in non-backwards-compatible ways, impacting dependent software that migrates to the new version [Pancake, 1995; Perkins, 2005; Xing and Stroulia, 2007b]. Because the source of the change is external to their software, developers are not able to plan alternative implementations to mitigate its impact: instead, they must determine how to the redress the impact of the library's change on their software [Dagenais and Robillard, 2011].

Early approaches focused on identifying cases where functionality was moved elsewhere in the API, suggesting how developers could redress their code [Godfrey and Zou, 2005; Kim et al., 2005], but this does not help when functionality is removed [Xing and Stroulia, 2007b; Dagenais and Robillard, 2011]. API change recommendation approaches address these cases by using varying heuristics to suggest similar alternatives in the API [Xing and Stroulia, 2007b], examining the library's implementation for examples of how it migrated its usage of the API [Schäfer et al., 2008; Dagenais and Robillard, 2011], or combining multiple approaches into a hybridized recommender [Wu et al., 2010].

Unfortunately, assessments of these approaches are limited, as there are no complete sets of library change data on which to evaluate these techniques. Without *a priori* knowledge of all library changes, researchers need either to restrict their evaluations to a few changes at the expense of generalizing to the rest of the library [Balaban et al., 2005; Kapur et al., 2010; Nita and Notkin, 2010; Dagenais and Robillard, 2011]; or to manually investigate their tool's predictions for correctness, accepting that they cannot know what changes their technique failed to recall [Kim et al., 2007; Xing and Stroulia, 2007b; Schäfer et al., 2008; Wu et al., 2010]. Not only do these limitations leave us with an incomplete picture of their effectiveness, it hampers identifying those areas for which additional research is needed.

The lack of publicly available data on how software libraries change is reflected by how few empirical studies exist on library evolution and API migration. Only one study of note has attempted to identify breaking-changes between API versions [Dig and Johnson, 2006a], but limited the scope of their inquiry to changes that they could detect through documentation and heuristics and to changes that they considered intended for reuse. Another study provided evidence that libraries change frequently and severely over their lifetime, but did not attempt to recover the changes in each version [Kapur et al., 2010]. Addressing the shortcomings in these works will be necessary prior to attempting to tackle further research on the problem of library migration.

1.2 Semi-Automating Developer Configured Solutions

The challenge of conducting dependency analysis in polylingual systems, and the difficulties in finding alternatives to altered or removed API functionality in external libraries, are different facets of the same problem.

- A developer *proactively* investigating the impact of changes to a polylingual system knows the source of the change, knows what the intended change will be, and will normally find the task of determining how to redress dependent code to accommodate their planned changes a straightforward one. Where she needs help is in finding code dependent on the locations she needs to change.

- A developer *reactively* redressing the impact of changes from an external library on their software system knows the source of the change (in the API), but does not know what the change was, and consequently will not know how to redress his dependent code. But she can readily find the areas in his software that are affected by these changes, simply by incorporating the new library version into his software system, and recompiling.[1] Where she needs help is in finding how the API has changed, and if necessary, finding alternatives for removed functionality.

Providing help in these cases through general-purpose, fully automated tool support has so far proved unsuccessful, or has only limited evidence as to its effectiveness. The problem is that the complexities of these analyses are difficult to mitigate, and automating their analysis has required overly specialized solutions that restrict the changes handled. At the opposite end of the spectrum are techniques requiring complete, detailed specifications by the end-user developer to effectively construct their own analysis tool; a task so burdensome that often largely manual investigations are seen as less work.

[1] For the purposes of this work, I am not interested in the problem of behavioural changes made to functionality in a library's API, where the API itself is unaltered, leading to breaking changes that are not detectable at compile time, but only later during the software's execution. While this too is a problem, it is not what I am interested in addressing.

In this work, I propose using the paradigm of *semi-automated* approaches for generating tool support for software developers in change impact analysis and redress tasks. Semi-automated approaches require that the developer provide some form of specification or training feedback to a tool that can then create or adapt its analysis to the specific characteristics of the developer's problem. Such configurations can be iteratively refined through additional or revised specifications, or further training feedback until the tool's analysis is sufficiently correct for their needs.

A semi-automated approach to proactive and reactive investigation of change impact and redress can offer several advantages over fully-automated, and non-automated solutions. A tool developer does not need to anticipate every language, technology, or possible analyses that may be needed by the tool's end-user, but can instead provide a general analysis framework that the end-user tailors to their context. Further, such tools have alternatives for problems where the necessary analysis is difficult to implement or computationally expensive, but can be reasonably thought through by a professional developer, by explicitly asking for feedback and input in the process of solving the problem [Cottrell et al., 2008; Kapur et al., 2010]. Unlike manual solutions, the burden is not placed entirely on the end user: an appropriate choice of how configuration information is provided, such as a notation language tailored to the characteristics of the analysis at hand, or the use of feedback mechanisms to provide training data, eases the difficulty applying such support to the developer's task.

I apply semi-automated approaches to the problem of change impact and redress in two ways.

1. For proactive investigation of change impact in polylingual systems, I provide a simplified means for developers to describe, or "sketch", what the key cross-language dependencies look like in their software system, to configure a customized dependency analysis tool. Developers can explore the detected dependencies, and refine their sketches until the configuration is sufficiently effective for completing their impact investigation task. This approach has the added advantage of allowing per-language sketches to be shared and reused as appropriate in other systems to speed

up the development of new configurations.

2. For reactive investigation of change impacts caused by software library evolution, and API migration, I hybridize a series of API change recommendation (which individually fare poorly) by requiring the end-user to provide examples of correct adaptations for API change; this training data is used to configure a rule-based machine learning algorithm to analyze their software library migration. Unlike previous attempts at hybridization, the combination of analyses is flexible and can be expanded or restricted by the end-user as desired, and the decision process for combining their results into a single prediction is not fixed in advance by a tool developer, rendering it possibly unsuited for an API with different change characteristics.

1.3 Thesis Statement

The thesis of this work is that *semi-automated approaches to change impact analysis and change recommendation systems provide more effective tool support than the fully-automated or manual alternatives available*.

By "more effective", I intend different things in different contexts but these always center on the idea that there are desirable results (such as the actual dependencies to be detected, or correct replacement functionality) that depend on the task at hand and undesirable results (anything else). In qualitative studies, the phrase indicates a reduction in effort to configure and use the novel tool support as well as an increased tendency to find desired results and to not find undesired results. In quantitative studies, the phrase indicates an increase in production of desired results and a decrease in production of undesired results; that is, some false negatives produced by comparable approaches become true positives in the novel tooling, and some false positives become true negatives. Quantification of the meaning of "more effective" is examined through a variety of measures in an attempt to avoid criticism that some particular measure might misrepresent the truth.

1.4 Organization

The dissertation continues in Chapter 2 with the presentation of two examples intended to motivate the problems of proactive and reactive change impact analysis and redress, by illustrating the limitations of current practice, and in Chapter 3 I present the related work. Chapter 4 presents my research into semi-automatic dependency analysis tool support for polylingual software systems, and explains how we designed the DSKETCH prototype analysis tool to use simple developer-provided specifications to generate a customized dependency analysis tool that was configured for their particular software system.

In Chapter 5, I begin my presentation on the problem of redressing API changes to developer source code with an empirical study on the kinds of changes software library APIs undergo between versions, and the effectiveness of techniques currently proposed in the literature for identifying the necessary repairs. Chapter 6 presents my evaluation of *matching via structural similarity* as a new means of inferring replacement functionality for missing or altered components of a software library, and presents the UMAMI prototype change inference tool. Chapter 7 expands on my previous findings by examining how existing proposed inference techniques, UMAMI included, can be flexibly combined into a single hybrid change recommender, by applying user-provided feedback on prediction correctness to train machine learning approaches to recognize how to best interpret the predictions of each recommender.

Chapter 8 discusses the results and findings of my work, as well as future work that I have planned. I conclude my dissertation, and present the original contributions of my work in Chapter 9. An overview of additional materials appended to the end of this dissertation is provided in Appendix A.

1.5 Publications

Two chapters in this work have been published in ACM conferences:

- Chapter 4 is a minor revision of the work published as "DSKETCH: Lightweight, adaptable dependency analysis" by Bradley E. Cossette and Robert J. Walker in the *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2010, DOI: 10.1145/1882291.1882335. ©2010 ACM, included here by permission.

- Chapter 5 includes a minor revision of the work published as "Seeking the ground truth: A retroactive study on the evolution and migration of software libraries" by Bradley E. Cossette and Robert J. Walker in the *Proceedings of the Twentieth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2012, DOI: 10.1145/2393596.2393661. This work was awarded an ACM Distinguished Paper Award, at the ACM SIGSOFT International Symposium on Foundations of Software Engineering, Carey, North Carolina, November 2012. ©2012 ACM, included here by permission.

Appropriate notices of copyright and permission to reuse these works have been noted in the chapters in which they appear, as well as in Appendix C.

Other portions of the thesis will be published, possibly with extensions, in the future.

Chapter 2

Motivation

> The business changes. The technology changes. The team changes. The team members change. The problem isn't change, per se, because change is going to happen; the problem, rather, is the inability to cope with change when it comes.
>
> Kent Beck [Beck and Andres, 2004, p. 28]

To illustrate the challenges that lie in effectively performing change impact analysis and redress in software systems, I present two scenarios involving a hypothetical software engineer, Tina, who is making changes to or reacting to changes in software systems for which she is responsible. First, I present an example of Tina proactively investigating the impact of considered changes to a software system; this is complicated by the intertwining of multiple languages and technologies that obscure what otherwise should be simple dependency relationships (Section 2.1). I then present how she approaches changes initiated by external sources, examining how she determines what possible forms of redress may exist (Section 2.2).

2.1 Proactive Change: Polylingual Dependency Analysis

Consider a scenario in which Tina is asked to create an online web-store for her company's employees to access over an internal network. The intent is to allow employees to browse and order company-branded clothing and merchandise, either as a reward for service or at a subsidized rate to promote company loyalty and identity. Because this is a small, internal application, Tina believes that it might be worthwhile to adapt an existing system to her needs, specifically the IBATIS JPETSTORE v5.0 reference application[1] (a small-scale web-store system that ships as an example with the IBATIS Object/Relational Model framework). She knows that the login mechanism used

[1] http://ibatis.apache.org

by JPETSTORE will need to be replaced with one that leverages the company's existing network login credentials for each employee. Thus, she must estimate the impact of this change on the rest of JPETSTORE.

JPETSTORE uses a series of XML files to define an object–relationship mapping between JAVA classes and SQL queries, for which the IBATIS framework can then generate JAVA objects at run-time. Using a text editor, Tina looks for the sql−map−config.xml file, which defines which XML mappings are in use, and decides that Account.xml may be the most relevant. Browsing the queries in the file, she notices that account information is stored in the SIGNON table, which has two columns: USERNAME and PASSWORD. One of the relevant SQL queries for adding a new user's credentials (from Account.xml) is presented in Listing 2.1.

```
1 <sqlMap namespace="Account">
2   <typeAlias alias="account" type="com.ibatis.jpetstore.domain.Account"/>
3     <insert id="insertSignon" parameterClass="account">
4       INSERT INTO SIGNON (PASSWORD,USERNAME) VALUES (\#password\#, \#username\#)
5   </insert>
6 </sqlMap>
```

Listing 2.1: Account.xml

As a first attempt, Tina runs a case-insensitive lexical search on the entire system to find other references to the USERNAME column; the search returns 106 matches. Looking through the results, she believes that these are mostly false positives so she fails to examine them closely (as a result, she overlooks references from within the Order.xml file). She restricts the search to just the Account .xml file: three other queries that access the USERNAME column are found. Tina is reasonably confident that these are the only three queries that she has to worry about, as the names of the other XML files do not suggest that they deal with user authentication. Her next step is to identify what JAVA functionality is dependent on these queries. To do this, Tina must understand the semantics of the IBATIS framework in mapping a query to a JAVA method. As an example, in Figure 2.1 the value of the id attribute in the insert node is the same as the first parameter in update("insertSignon ", account), a method invocation inside the insertAccount() method in the AccountSqlMapDao class.

The class extends part of the IBATIS framework, and uses the update method it inherits to perform the query lookup at run-time. She determines that a total of four methods in the AccountSqlMapDao class are dependent on the authentication queries (two getAccount(..) methods and insertAccount(..) are the others).

To continue her feasibility study, Tina determines what other parts of the system are dependent on these four methods in the AccountSqlMapDao class. After building the JPETSTORE system in an integrated development environment, such as ECLIPSE[2], she can use the semantic tools it provides to quickly infer what classes are dependent on these methods. She determines that three classes are dependent on the four AccountSqlMapDao methods: Account, AccountBean, and AccountService.

Tina now has a problem: she knows that several web-pages (forming the user interface for this system) accept login information, and likely interact with the JAVA code. But unlike the XML files, there is no contextual information in the JAVA code to suggest which pages those are, or on which entities in the JAVA code the functionality depends. Tina resignedly sets up a TOMCAT server on her machine, deploys the JPETSTORE system, and manually investigates the webpages. She guesses that NewAccountForm.jsp, EditAccountForm.jsp, and SignOn.jsp are the most relevant. These webpages are written using JAVA SERVER PAGES (JSP), and make use of yet another framework called APACHE STRUTS[3] to dynamically generate content.

Getting annoyed at the time this "estimate" is taking, Tina opens the three webpages in her IDE. Again, an understanding of how the STRUTS framework encodes dependency information is needed for her to manually and precisely determine what dependencies exist between each JSP page and the JAVA source code. In Figure 2.2, Tina knows that the name attribute refers to a JAVA object, and property refers to a field on the same object. Using a case-insensitive lexical search, the developer may correctly infer that "accountBean" is referring to the AccountBean class, which she identified earlier as depending on the authentication queries. But a lexical search on the fields within AccountBean will not yield a match for username or password. Tina must remember that the

[2]http://www.eclipse.org
[3]http://struts.apache.org

```
<table>
  <tr>
    <td>User ID:</td>
    <td>
      <html:text name="accountBean" property="username"/>
      </td>
  </tr>
  <tr>
    <td>New password:</td>
      <td>
      <html:password name="accountBean" property="password"/>
    </td>
  </tr>
</table>
```

Listing 2.2: NewAccountForm.jsp

STRUTS framework supplies "TagLibs" which allow an implicit mapping of the property attribute here to get/set *method* invocations on the AccountBean class (e.g., getUsername, setUsername), and alter her search accordingly. She must now repeat this manual analysis on the other two webpages before coming up with an estimate of the change impact on the JPETSTORE system in replacing the login mechanism.

2.2 Reactive Change: API Migration and Redress

For the sake of continuity in our example, let us suppose that Tina successfully develops her company's internal online webstore. Later, she finds that the Apache organization has released a new version of the APACHE STRUTS library; recall from Section 2.1 that during her investigation of the JPETSTORE system that she hoped to reuse, Tina discovered that STRUTS was one of the libraries used in its construction, and she realizes that the online web-store is now using an outdated version of that library. She may want to incorporate the new version into her system to take advantage of new features for future development, as well as enhancements or bug fixes to what her software is already using.

In the ideal case, the new library is *binary compatible* with the previous version: the library's application programming interface (API) has not altered significantly, and the JAVA virtual machine

can use the new compiled version of the library without having to recompile or otherwise modify existing calls to the library's API [Gosling et al., 2012, ch. 13]. This ideal is not always met: if Tina were using STRUTS v1.0.2, and was trying to migrate to the next version (STRUTS v1.1.0), she may see a series of compilation and class loader errors when she incorporates the new library version into her system, as over 200 *binary incompatible* (BI) changes were made to the classes, methods, and fields that comprised the STRUTS API.

To understand some of the challenges even simple API changes can pose, consider one such BI from the STRUTS v1.0.2 to v1.1.0 migration: org.apache.struts.action.ActionServlet.processLocale(), presented in Listing 2.3. The processLocale() method is not especially complex, but it is missing from the ActionServlet class in the v1.1.0 API. Unfortunately for Tina, the STRUTS development team did not provide any warning that this method was going away, or how it should be replaced: there is no mention of it in the release notes,[4] the change was not marked as deprecated in STRUTS v1.0.2, and the source code documentation does not say how to repair references in her code to it. While it looks as if this functionality is removed without a trace, research (and perhaps even Tina's own experience) shows that in most cases, equivalent functionality exists in the new API version, and its removal is an accidental byproduct of the refactorings or other changes that have caused its relocation elsewhere in the API [Dig and Johnson, 2006a; Xing and Stroulia, 2007b; Cossette and Walker, 2012].

Listing 2.4 shows the replacement for ActionServlet.processLocale() in STRUTS v1.1.0. In the new library release, the STRUTS developers decided to extract a number of HTTP request processing methods from the ActionServlet class into a separate class that could be easier to extend and override. The RequestProcessor class was introduced in version 1.1.0, and the processLocale() method was moved into it (among others). At first glance, it may not seem like much has changed: the method name is the same, the implementation is largely the same, and even the associated code comment is just a rewording of the original. But these minor changes can frustrate the inference techniques [Robillard et al., 2013] used to detect API migrations.

[4] http://struts.apache.org/release/1.2.x/userGuide/release-notes-1.1.html

```
1  /**
2   * Automatically calculate an appropriate <code>java.util
3   * .Locale</code> for this user, and store it in their
4   * session, if there is no such Locale object present
5   * already.
6   * @param request The request we are processing
7   */
8
9   protected void processLocale(HttpServletRequest request) {
10   if (!locale)
11     return;
12
13   HttpSession session = request.getSession();
14   if (session.getAttribute(Action.LOCALE_KEY) != null)
15     return;
16
17   Locale locale = request.getLocale();
18   if (locale != null) {
19     if (debug >= 1)
20       log("Setting locale " + locale );
21
22     session.setAttribute(Action.LOCALE_KEY, locale);
23   }
24 }
```

Listing 2.3: ActionServlet.processLocale()

For instance, suppose Tina starts by trying to employ lexical analysis to find equivalent functionality. She could try searching on the fully qualified name of the ActionServlet.processLocale() method, and look for similar methods in the new API based on the edit distance between their signatures. Doing so will present some initially promising candidates that after investigation will prove to be unuseful; in fact, RequestProcessor.processLocale() will not even show up in the top 100 matches, as there are many other methods that contain a higher degree of lexical similarity. The new class name and the additional parameter in the method signature throw off this particular technique. Unfazed, she could repeat the search, but this time on the associated code comments, hoping that the developer had simply copied the original code comment. Her results will be worse than before: a developer took some time to update the code comment, and while it basically states the same description, it is dissimilar enough to the original comment that hundreds of other obviously unrelated results will show up in her search. Tina could investigate the STRUTS v1.0.2 library implementation to see if it made use of the processLocale() method anywhere, in case it

```
1  /**
2   * Automatically select a Locale for the current user, if
3   * requested. <strong>NOTE</strong> - configuring
4   * Locale selection will trigger the creation of a new
5   * <code> HttpSession</code> if necessary.
6   * @param request The servlet request we are processing
7   * @param response The servlet response we are creating
8   */
9  protected void processLocale(HttpServletRequest request, HttpServletResponse response) {
10   if (!moduleConfig.getControllerConfig().getLocale())
11     return;
12
13   HttpSession session = request.getSession();
14   if (session.getAttribute(Globals.LOCALE_KEY) != null)
15     return;
16
17   Locale locale = request.getLocale();
18   if (locale != null) {
19     if (log.isDebugEnabled())
20       log.debug(" Setting user locale " + locale );
21
22     session.setAttribute(Globals.LOCALE_KEY, locale);
23   }
24 }
```

Listing 2.4: RequestProcessor.processLocale()

also migrated its dependency on that method, and thus would provide an example of the correct migration. Initially, she will have success: Tina will find one reference in the process() method of the same containing class, ActionServlet. However, when she compares the method's implementation to the same method in STRUTS v1.1.0, she will find that the entire implementation—88 lines of code—has been replaced by two method calls, whose implementations show no reference to anything resembling the missing method. At this point, Tina may rightly fear that the method has been removed without any replacement and that further searching is likely more work than implementing a workaround.

Tina would no doubt be frustrated to learn that the replacement functionality she is looking for has so much in common with the original, deleted method. Figure 2.1 shows an altered formatting of Listing 2.3 and Listing 2.4, where functionality common to both methods is aligned. As you can see, line 5 tests a boolean value, but retrieves it via different mechanisms; line 16 checks to see if a debug mode has been enabled, but a different test is used in each method; and line 20 tries to

```
1  protected void processLocale(
2          HttpServletRequest request) {
3
4
5    if (!locale)
6
7      return;
8
9    HttpSession session = request.getSession();
10   if (session.getAttribute(
11         Action.LOCALE_KEY) != null)
12     return;
13
14   Locale locale = request.getLocale();
15   if (locale != null) {
16     if (debug >= 1)
17       log("Setting locale "
18         + locale );
19
20     session.setAttribute(
21         Action.LOCALE_KEY, locale);
22   }
23 }
```

```
1  protected void processLocale(
2          HttpServletRequest request,
3          HttpServletResponse response) {
4
5    if (!moduleConfig.
6      getControllerConfig().getLocale())
7      return;
8
9    HttpSession session = request.getSession();
10   if (session.getAttribute(
11         Globals.LOCALE_KEY) != null)
12     return;
13
14   Locale locale = request.getLocale();
15   if (locale != null) {
16     if (log.isDebugEnabled())
17       log.debug(" Setting user locale "
18             + locale );
19
20     session.setAttribute(
21         Globals.LOCALE_KEY, locale);
22   }
23 }
```

Figure 2.1: Comparing ActionServlet.processLocale() to RequestProcessor.processLocale()

store the user's locale into the session's attributes, using a different named constant. The *structure* of the method is largely the same between versions: the same logical tests, object instantiations, and even method invocations are present, but details in each have changed. A more useful search for Tina might have tried to look for methods that had these structures in common, determined just how far apart these implementations actually were, and suggested which were the methods most similar in overall structure to her missing method. But all that Tina can do is use what's available to her.

2.3 Summary: Motivation

In both of these scenarios, the core problem that Tina faces is the difficulty of either understanding what needs to change in her software to accommodate a *proposed modification*, or understanding what has changed and why so that she can accommodate a *proposed migration*. The actual difficulty of the change, at least conceptually, is quite low; however, complexity is introduced into this

problem such that it is challenging to undertake without appropriate tool support.

In Section 2.1, what Tina really wants to do is select a few database queries that she thinks will change, and to quickly see a set of classes and webpages that could be affected. Because this is an exploratory investigation, some inaccuracy would be acceptable as she can either do some additional manual investigation to correct this, or will be fine with a close approximation. What ends up happening is that Tina needs to trudge through a largely manual, method-level investigation of the code with tool changes, context switches, and even a system deployment becoming obstacles to obtaining her estimate. Even though JPETSTORE is a small software system, the problems Tina has in investigating changes to it are not unique: even conceptually-small change tasks, when undertaken in the context of a polylingual system, can require demanding dependency investigation across multiple programming languages, protocols, and technology platforms.

Once Tina solves this issue, she faces a different challenge in Section 2.2, when she tries to migrate to a newer version of a library her system relies on, only to find some classes, methods, and fields that were part of the library's public API appear to have been removed. To successfully migrate her system to the new library version, Tina must determine how to redress the impact of these API changes on her software system; a problem amplified by the lack of information on how the API has changed, and what the replacement functionality might be. In fact, the example in Section 2.2 suggests that at least sometimes, there actually is a new method in the API which is a suitable or intended replacement which could easily be reused, provided a developer can find it. As the series of failed investigations show, despite the missing and replacement method having so much of their implementation in common, it can be actually quite difficult to discover that these are equivalent when searching through the sea of other possibilities in a large API. Tina simply has to try different ways to search for this related functionality until she stumbles upon it, or simply gives up and tries to remove her reliance on that part of the API.

Chapter 3

Related Work

> Software engineering is the part of computer science which is too difficult for the computer scientist.
>
> Friedrich L. Bauer [1971]

Research has recognized that change tasks on evolving and aging software systems are complicated by the dependencies present, and has long advocated for approaches to identify both the impact of changes and mitigation strategies [Moriconi and Winkler, 1990; Krueger, 1992]. Two areas are of particular interest to this work: (1) supporting dependency analysis in source code for identifying change impact; and, (2) identifying adaptations for redressing change stemming from external entities. Both problems are commonly faced by developers, tool-based approaches have been put forward for each. In this chapter, I discuss how existing research has addressed these problems, and discuss the challenges posed to many of the automated approaches available thus far.

I begin in Section 3.1 by introducing the dependency analysis side of the software change impact and redress problem, and discuss efforts made in supporting developers in their investigation of dependencies. In Section 3.2 I address the particular challenges now faced in conducting dependency analysis in the context of software systems developed using multiple languages and technologies. In Section 3.3 I discuss the redress side of the change impact problem, focusing on the challenges of changes emanating from software library evolution, and discussing the approaches that aim to recommend adaptive changes to redress the impact of API changes. Finally, in Section 3.4 I present the available empirical research on how software libraries and their APIs change over time, and its important for which is important as a foundational understanding as the implications for our understanding of how well available approaches can actually redress the impact of software change.

For the sake of the reader (and in recognition of the length of some sections), summaries are provided at the end of each section containing a brief overview of the key themes for that portion of the related work. A summary of the entirety of the chapter is provided in Section 3.5.

3.1 Change Impact and Dependency Analysis

Brooks [1987] ascribed problems in software development as being *accidentally* complex, or *essentially* complex, and this paradigm is useful for understanding how different lines of research have treated the problem of software change impact. Early research in change impact viewed this as a serious, but accidentally complex problem: that is, that choices made during the design and modification of the software were largely responsible for future changes having inordinate impacts [Parnas, 1972; Stevens et al., 1974; Belady and Lehman, 1976]. Consequently, a number of software engineering design principles were proposed, based on providing different forms of abstraction with the purpose of either removing unnecessary dependencies, or structuring dependency relationships to easily accommodate future modifications, thus preventing change impact [Parnas, 1972; Stevens et al., 1974; Parnas and Siewiorek, 1975; Parnas, 1979]. To some degree, the problem of change impact in systems was seen as solvable through appropriate design: while it was recognized that future changes could not be perfectly anticipated, and thus a perfect abstraction that accommodated all changes was impossible, the initial response to this was to try and delay key decisions until more was known [Wirth, 1971].

Brooks [1987] however notes that the complexity of software systems, and their need to change over time, are key examples of the essential complexity, or inherent difficulties, present in software design and development [Brooks, 1987]. Successful software systems often outlast the hardware platforms they were originally written for, and users find new problem domains for which the system could be useful, none of which can be foreseen by the developers during its initial design [Parnas, 1972; Brooks, 1987; Parnas, 1994]. The consequence is that the nature of some changes will not be containable to the abstractions put in place to insulate the software from future changes,

and their impact may ripple across the new network of dependencies in the software in negative ways [Podgurski and Clarke, 1990; Arnold and Bohner, 1993]. With this came the recognition that it was not possible for developers to anticipate all future changes in their designs: some forms of analysis would be needed to assist developers in resolving change impact when it happened.

3.1.1 Formulating Change Impact

Because software change was recognized as unavoidable, and that the impact of these changes could not be prevented solely through better design, research began to address the problem of identifying what the impact of changes could be, in the hopes that this would allow developers to mitigate them [Luqi, 1990; Moriconi and Winkler, 1990; Podgurski and Clarke, 1990; Landi, 1992]. While some approaches still took the mindset that the problem of change impact could be carefully managed and controlled [Luqi, 1990], other lines of research focused on how to assist developers identify and mitigate the consequences of their changes. Much of this early work was highly formal, and attempted to clearly identify when a dependency relationship existed between two entities in software, but it was recognized that approximations of dependency relationships were ultimately necessary. For example, Podgurski and Clarke [1990] note that the core problem of dependency analysis, namely if a change to one entity in the software system will affect the behaviour of any other entity, is undecidable. To overcome this, they introduce the idea of "semantic dependency" between two program statements as occurring when one statement's correct functioning relies in part of the correct functioning of the other; by implication, a change to one statement can only potentially affect those statements that are semantically dependent upon it. To identify these semantic dependencies, Podgurski and Clarke apply a series of syntactic analyses aimed at identifying control- and data-flow dependencies between entities, which are then used to identify if the conditions for a semantic dependency between those entities are met. Even here though, Podgurski and Clarke must qualify that their syntactic analyses only indicate that a semantic dependency could exist.

Moriconi and Winkler [1990] provide a conceptually similar approach to identifying dependencies by looking for "information flow" between entities, an approach analogous to data flow analysis by Podgurski and Clarke [1990]. However, Moriconi and Winkler are explicit in noting the value of detecting where dependencies *could* exist between entities, rather than trying to clearly identify that they do. By overestimating the potential locations where dependencies may exist, the developer is assisted by being certain that they have not overlooked any potential points of impact, at the expense of needing to investigate entities that may prove to be unaffected. While not explicitly stated, the work implies that creating extra work for a developer through overly conservative dependency detection is preferable over the alternative: a change introducing a defect because an affected dependency was never identified.

The dependency analysis proposals of both Podgurski and Clarke and Moriconi and Winkler are essentially forms of *static* dependency analysis: that is, the analysis is based on the lines of code comprising the software's implementation. Thus, each approach relies, at least in part, on its ability to analyze the implementation details of the software to determine how data or information flows between various points in the software, and thus estimate where dependencies exist. However, Landi [1992] later showed that even these estimates were not complete, in that despite the argument that these techniques would identify all potential points of dependency through conservative analysis, the static dependency analysis relied upon was itself an undecidable problem. Landi argue that, because programming languages offered the opportunity to define different *aliases* that all referred to the same storage location in memory, two problems exist: knowing which aliases all refer to the same point in memory, and also knowing *when* each refers to those points, as aliases themselves can change over time. While previous research suggested this was a hard problem that could be successfully approximated, Landi showed that this in fact was an undecidable problem; by extension, the work shows that static dependency analysis is also undecidable, and thus cannot be guaranteed to find all possible points of dependency in software.

Despite these limitations, research on how to determine dependency relationships for the analysis of change impact proceeded apace, and Arnold and Bohner [1993] note that there was at the time a lack of consensus on what change impact analysis actually was, what it consisted of, and how it needed to be applied. As part of trying to clearly define the practice, Arnold and Bohner introduce three terms useful for discussing what change impact analysis is. A change to the software begins with developers identifying a *starting impact set* (SIS), which is the set of entities they initially expect that will need to be modified as part of implementing a proposed change. They then explore dependency relationships in the software to identify an *estimated impact set* (EIS), or those entities which may be affected by this change to the software. Once an initial estimate is made, developers use this information to refine the planned change, perhaps adjusting its scope in recognition of the additional work needed. To evaluate the effectiveness of various change impact analysis approaches, Arnold and Bohner define a third set of entities, the *actual impact set* (AIS) which consists of the complete set of all entities which must be modified in some fashion to correctly enact the change.[1] The relationship between entities contained in either the AIS or EIS measures the effectiveness of the impact analysis in that case: as with Podgurski and Clarke [1990] and Moriconi and Winkler [1990], the goal is that the AIS be completely contained within the EIS, and ideally that there be few to no entities in the EIS that are not part of the AIS. Arnold and Bohner [1993] also acknowledge the risks pointed out by Landi [1992] by discussing the case where the EIS does not encapsulate all the entities in the AIS, or worse, has no entities in common.

3.1.2 Reverse Engineering and Tools For Dependency Analysis

In parallel to the work on formal models, tool-based approaches to assisting developer investigation of change impacts began to appear in the area of reverse engineering. Adapted from the hardware practice of discovering the design from the implementation, software engineers use reverse engi-

[1] Arnold and Bohner note that the concept of a true actual impact set is "fuzzy" in that there may be multiple correct ways to implement an actual change. For conceptual purposes, they instead assume that the AIS for a given change is the complete list of entities that need to be modified to enact the change as the developer has planned. They also assume that the AIS represents a correct modification to the system, rather than a partial modification which leaves out necessary updates.

neering to create abstract views, or models of the software's implementation that are otherwise difficult to determine from direct perusal [Chikofsky and Cross, 1990; Luqi, 1990]. To accurately analyze the source code underlying the software's implementation, such tools were constructed to be *semantically-aware*; that is, the tool understood the syntax and the semantics of the programming language used in the source code [Cossette and Walker, 2007]. To facilitate this awareness, the tool's creator would create (or reuse) a parser for the programming language, and create a semantic analyzer that could analyze the resulting parsing tree to construct the particular model(s) for which the tool was designed [Reps and Teitelbaum, 1984; Cleaveland and Kintala, 1988; Chen et al., 1990; Grass and Chen, 1990; Reubenstein et al., 1993; Devanbu, 1999].

A common application of reverse engineering was attempting to model dependency relationships in the software by *cross-referencing* references to functionality in source code against their points of declaration, effectively a simpler version of the syntactic analysis ideas put forward by Podgurski and Clarke [1990] and Moriconi and Winkler [1990]. A typical example is the C INFORMATION ABSTRACTION (CIA) system of Chen et al. [1990], which focused on cross-referencing identifiers in C source code that had significance beyond their immediate location. Their approach involved writing a parser to extract identifiers that could have global significance in the software (in the C programming language, this included function names, macros, types, header files, etc.), and explicitly cross-referencing the declaration of these identifiers against their occurrences in the software. This information was stored in a database that software developers could later write queries against: for example, a developer needing to change a function could write a query to identify all references to that function in the software, which would be a useful as a starting estimate on the impact of the change.[2] Today, such cross-referencing capability is standard functionality in integrated development environment (IDE) tools for software development.[3]

[2]Surprisingly though, this application of cross-referencing escaped the authors: Chen et al. [1990] cite program understanding, reuse extraction, and program restructuring as primary applications of their work. While they mention the value of detecting entities which share a high degree of dependence, they interpret this as being useful only for either recovering design decisions or planning future restructuring to eliminate those dependencies.

[3]For example, in ECLIPSE: http://www.eclipse.org

3.1.2.1 The Cost of Semantically-Aware Tools

Despite the power and usefulness of this new wave of reverse engineering tools, it was noticed early on that they suffered from requiring detailed syntactic and often semantic understanding of the languages they analyzed in order to be effective, and as such the provided tools were only useful for their target language. As new languages were introduced, researchers found that modifying these tools to support additional languages was quite difficult, and prone to limitations.

In a follow up to Chen et al.'s work on the CIA system, Grass and Chen [1990] noted the lack of development tools to support C++ development despite its popularity, and found that this was because tools for reverse engineering C code did not transfer gracefully to reversing C++ code, despite the high degree of commonality in syntax. To get around this, Grass and Chen [1990] used the CFRONT compiler [Stroustrup, 2007][4] to convert C++ code into C code, which then could be fed into their existing CIA tool for analysis. But this raised several issues: CFRONT's compilation process into C code sometimes destroyed or rearranged information in ways that could not used by the CIA tool; Grass and Chen also found it difficult to modify the CFRONT compiler to extract needed information for their analysis as well, and thus needed to limit their tool's capabilities. Reubenstein et al. [1993] also tried to add additional languages and new analyses to existing reverse engineering tools, and found it problematic: while it seemed sensible to write an additional parser for the language and integrate that into the system, they found that the tools they sought to modify had tightly integrated the language syntax and semantic processing, such that it was difficult to incorporate additional syntax into the same semantics. Furthermore, Reubenstein et al. noted that the structure of the grammars in general made it difficult to conceptually understand how the parser worked, and how to collect the semantic information necessary for analysis: because a hierarchy of rules is typically used to define a grammar, the information needed for semantic analysis is scattered.

Some approaches considered implementing an additional abstraction layer between the language representation in each tool, and the analysis engine, to allow differing languages to be

[4]Because CFRONT converted C++ code into C code, it would also be fair to call it a preprocessor.

united under a common representation, provided a parser could construct the appropriate semantic representation. Reubenstein et al. [1993] proposed identifying common semantic constructions in various languages (e.g., what is the equivalent of a "procedure" in each language). Effectively, new parsers would be written to parse source code into this language-independent format; however, their approach was by their own admission not able to handle some semantic constructions which were unusual to one language, raising questions as to whether this would prove to be a common problem [Reubenstein et al., 1993, p. 121]. While Reubenstein et al. began their work warning that adding another parser was not always the solution in these cases, their approach fundamentally requires the creation of new parsers for every language [Reubenstein et al., 1993, p. 117].

A similar idea lies behind the GENOA system [Devanbu, 1999], created to address the problem of re-targeting reverse engineering tools to new languages. Instead of defining common semantic concepts in languages, GENOA has a subsystem called GENII which defines a simplified abstract syntax tree (AST) representation of source code that was hoped to be universal across programming languages; thus, the various reverse engineering analyses developers designed could be written against the universal AST, rather than a specific programming language, and thus be broadly useful others needing re-implementation. The catch, as it were, is translating source code effectively into this AST. GENII requires that someone write a language specification in its proprietary format to describe how a programming language's syntax maps into the universal AST: effectively, returning to the problem of "write another parser" [Reubenstein et al., 1993]. Devanbu [1999] note that writing language specifications for GENOA can be hard, particularly for certain languages, but do not discuss how difficult it is to adapt these specifications to changes in the language, or for other dialects [Reubenstein et al., 1993; Moonen, 2001, 2002; Stroustrup, 2007]. Furthermore, Devanbu [1999] note that the loss of some kinds of syntactic information in the GENII universal AST would limit the kinds of semantic analyses that developers could apply in the GENOA tool, a similar problem faced by Reubenstein et al. [1993].

3.1.2.2 Syntactically- and Lexically-Aware Tools

Implementing complete semantic support for new languages or their dialects to existing reverse-engineering tools by writing and integrating new parsers was proving to be a difficult and costly approach in many cases [Grass and Chen, 1990; Reubenstein et al., 1993; Devanbu, 1999]. However, researchers began to find that complete semantic support was not always necessary; often, only a subset of a language's syntax and semantics needed to be recovered to successfully model some aspect of interest [Cossette and Walker, 2007]. Such tool support would not need to be aware of all of a language's syntax or semantics; instead, tools could be *syntactically-aware* (i.e., understanding the language's syntax, but relying on the end-user to provide semantic analysis), or *lexically-aware* (i.e., having no understanding of a language's syntax or semantics, by instead relying on lexical pattern matching).

For example, Wong et al. [1995] described a case in which they attempted to apply their RIGI tool to reverse-engineer the architecture of the SQL/DS system,[5] and from it generate documentation. One of the problems they encountered was that SQL/DS was written in IBM's PL/AS language; RIGI had parser support only for the C and COBOL languages. Wong et al. noted not only that incorporating PL/AS required them to write a new parsing subsystem for the RIGI tool, but that writing a parser for PL/AS was difficult. Instead, they observed that to identify high-level architectural details in the source code, only *some* of the language syntax was of importance, and this could be obtained by applying lexical matching and *regular expressions* to the source code. The extracted syntax could then be treated such that the existing C analyses available in RIGI could be used [Wong et al., 1995, p. 49]. Effectively, RIGI implemented lexically-aware tool support for the PL/AS language.

Lexically-based tools require the developer to define regular expressions to recognize dependency structures in code. Developers can then configure such tool support for the languages or technologies that are peculiar to the software systems that they wish to analyze. The most well-

[5] SQL/DS was an IBM database management system; for more information, please see http://en.wikipedia.org/wiki/IBM_SQL/DS

known lexical analysis tool is the GREP family of tools [Bourne, 1977], which provide support for matching regular expressions on any text source. A more refined example of lexically-aware tool support would be the work of Murphy and Notkin [1996] in creating the LEXICAL SOURCE MODEL EXTRACTION (LSME) tool. Developers define lexical patterns to match relevant source code using regular expressions, and also provide a second set of specifications to instruct LSME on what to do when a lexical pattern finds a match in the source code: effectively, a lexically-aware implementation of a parser and semantic analyzer. The instructions allow developers to combine matched patterns to create their own reverse-engineered models of the source code, rather than rely on what a tool may provide. LSME improved on several of GREP's limitations:

- LSME provides implicit tokenization of the source code: the tool clusters together any sequence of characters that are not separated by whitespace or special "escaped characters", and treats them as a single word. The developer can write lexical patterns to operate on these words, sparing developers from having to specify what constitutes an identifier for every language analyzed.

- The tool allows patterns to share a *hierarchical* relationship, such that a pattern may have a requirement that another pattern must first be matched in the source code before it may match anything. This allows developers some flexibility in constructing fine-grained patterns by combining smaller patterns together.

- LSME provides a separate mechanism for operating on matches found in the source code, allowing developers a means of combining lexical patterns in more sophisticated ways than GREP allows.

A syntactically-aware variant on the above would be the TAWK tool of Atkinson and Griswold [2006], which constructs an abstract syntax tree (AST) representation of source code using a provided (or developer implemented) parser, but then supports a regular-expression based analysis of the AST for modelling and analyzing the source code; effectively, a simplified version of the

GENOA tool [Devanbu, 1999]. To use TAWK, a developer starts by providing a grammar specification for the programming language used in their source code, generating a parser that will later construct an AST representation of that code. The developer then constructs their desired analysis by writing regular expression patterns to match the text contained in the AST; however, the patterns can be augmented with some additional qualifiers to take advantage of the known syntactic structure, and thus limit mistakes in the analysis due to lexically-similar but syntactically-different identifiers.

3.1.3 Summary: Change Impact and Dependency Analysis

Initially, research on the problem of change impact analysis focused on preventing modifications from requiring unplanned changes to dependent code through identification of design principles that emphasized anticipating and containing future changes. It was later recognized that not all changes can be anticipated, and modifying software is particularly at risk for introducing defects when it occurs later in the software's life cycle. Approaches were needed to identify the impacts of these later changes, so that they could be planned for and mitigated.

While much of the early work on change impact analysis was highly formal, it was quickly recognized that techniques for approximating the impact of changes were needed. Tool-supported dependency analysis techniques were provided by reverse-engineering the source code of a software system to model the dependency relationships present; such models made estimating the impact of future changes to the software easier than direct investigation of the code. However, early efforts in constructing these tools showed that semantically-aware tool support was expensive to construct, and adding support for new languages or dialects to the tool was problematic: differences in language structures could result in parsers that were difficult to incorporate into the existing semantic analysis framework.

Later work attempted to simplify the problem of adding additional language support, through the provision of an intermediary abstract syntax tree intended to be representative of all languages. However, other lexically-aware and syntactically-aware approaches began to emerge, based on

the recognition that not all of a language's syntax or semantics are necessary for some forms of analysis. Omitting these details allowed for a faster specification of analyses for new languages.

3.2 Polylingual Dependency Analysis

In Section 3.1.2 several problems were noted when existing tools for reverse engineering code, or more specifically for performing dependency analysis, needed to be applied to software written in another language. Researchers noted that adapting existing tools to accommodate new languages required extensive rewrites of semantic-analyzers, implementing new parsers, and in some cases required ignoring some semantic information that was too difficult to adapt the tool to capture [Chen et al., 1990; Grass and Chen, 1990; Reubenstein et al., 1993; Wong et al., 1995; Devanbu, 1999]. But increasingly support is needed not just for software systems written using new languages and dialects, but also for systems where multiple programming languages and dialects are used throughout the source code.

Such *polylingual* software systems are not new: embedding SQL query fragments inside code written in another language (e.g., JAVA or C++) to communicate with databases has been a programming practice for some time [e.g., Burson et al., 1990]. Like the JPETSTORE example from Section 2.1, the structure of these polylingual systems is roughly similar: a combination of programming languages is used to implement the software, typically to take advantage of each language's strength in differing problem domains, with additional technologies used to facilitate inter-language communication and interaction [Moise and Wong, 2005; Furr and Foster, 2006; Kontogiannis et al., 2006; Cossette and Walker, 2007]. For example, J2EE 1.4 used XML to supply additional dependency information for classes, such as embedding database queries using an SQL dialect that were mapped to methods in a class [Perrone et al., 2003].

Some protocols are independent of the implementing languages, such as web services,[6] and the COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)[7]. Others target specific

[6] http://www.w3.org/TR/soap

[7] http://www.omg.org/technology/documents/formal/corba_iiop.htm

language combinations, like the JAVA NATIVE INTERFACE which bridges JAVA and C/C++ code [Liang, 1999]. Still others rely on library frameworks to allow code from one language to be embedded as text, and interact with another language by supplying the code's text as an argument to the framework, such as is common with the JAVA DATABASE CONNECTIVITY API (JDBC),[8] or the JAVA SCRIPTING API.[9] The semantics describing the cross-language dependencies in each system are often specific to the technology used, and not generalizable.

This poses a problem for constructing tool support for dependency analysis: approaches that leverage existing reverse engineering tools for new languages and dialects did so to save the cost of constructing a new tool. While the costs of writing a new parser, or significantly modifying the semantic-analyzer were not insignificant, they were seen as less than creating a new tool [Chen et al., 1990; Grass and Chen, 1990; Reubenstein et al., 1993; Wong et al., 1995]. But supporting analysis in polylingual systems requires parsers for every language in use: in the example from Section 2.1, five programming or markup languages are in use. The semantics of how these languages interact are defined by one or more external technologies, rather than the languages themselves: again, prior example contained three different technologies facilitating cross-language dependency relationships (see Section 2.1). This specific combination of languages and technologies are not unique either, as there are several data-mapping technologies using JAVA, XML, and SQL to facilitate JAVA source code interacting with an object-based representation of a relational database. Thus, providing semantically-aware support for polylingual systems requires not only providing a parser for each language, but a separate semantic analysis engine for each dependency-facilitating technology present: a combinatorial explosion of configurations that cannot be supported in the general case [Biggerstaff et al., 1994].

The alternatives to supporting polylingual dependency analysis in these cases are either lexically-aware approaches that ignore details about the underlying language by instead operating on textual patterns, or syntactically-aware approaches that lack semantic understanding of the language.

[8] http://www.oracle.com/technetwork/java/javase/jdbc/index.html
[9] http://docs.oracle.com/javase/6/docs/technotes/guides/scripting/programmer_guide

3.2.1 Lexically-Aware Tools in Polylingual Contexts

Lexically-based dependency analysis tools permit the developer to define patterns, often based on regular expressions, to recognize dependency structures in code. While flexible and descriptive enough to be useful for polylingual systems, they possess significant usability issues in practice that result in their poor comprehensibility and modifiability.

```
1 ([a−zA−Z_]\w*) \s+ ([a−zA−Z_]\w*) \s* "(" \s*
2 (([a−zA−Z_]\w*) \s+ ([a−zA−Z_]\w*))? (\s* "," \s*
3 ([a−zA−Z_]\w*) \s+ ([a−zA−Z_]\w*))* \s* ")" \s*
4 "throws"? ([a−zA−Z_]\w*))? (\s* "," \s*
5 ([a−zA−Z_]\w*))\s* "{"
```

Listing 3.1: An arcane example of a regular expression

As an example, consider the regular expression in Figure 3.1, used to describe the start of a method declaration (not including modifiers) in JAVA. There are five errors in this expression that I know of, despite the fact that during its creation it was never intended to have any mistakes, and multiple sets of eyes had reviewed it over an extended period. These errors are likely not obvious to a developer under pressure to complete a task quickly.[10]

The LEXICAL SOURCE MODEL EXTRACTION (LSME) tool by Murphy and Notkin [1996], introduced in Section 3.1.2.2, improves on several of the limitations of regular expressions, and

[10]For the sake of completeness, the known errors are:

1. There is a `s+` before the opening "(" that indicates the start of the parameter list: this means there needs to be whitespace between the method name, and this parenthesis. That is not correct: the JAVA language specification allows there to be no space, and so this will reject a valid method declaration. As a note to posterity, it was Dr. Jörg Denzinger who pointed this out to me in a very memorable fashion during my Master's defense.
2. The expression to match the parameter list, between "(" and ")", will accept a parameter list that starts with a comma which is invalid syntax.
3. The expression to match the **throws** list will also accept a comma at the start of the list, which is invalid.
4. Any types (i.e., return type, parameter type) in the method declaration that have the [] syntax to indicate that they are an array will not be matched, even though that is correct syntax.
5. Finally, the regular expression has no support for generic types, although they were introduced years ago with JAVA 1.5.

With the latest release of JAVA (version 1.8) now offering support for lambda expressions, the pattern is further obsolete than I ever intended.

could be useful in polylingual systems. However, LSME has several significant limitations in the context of polylingual dependency analysis.

Because LSME is a lexical analyzer, it cannot leverage basic knowledge about the syntax of languages under analysis. While this ignorance allows it to be flexibly applied to any language, it also means that it cannot take advantage simple constructs a developer may know about that language. For example, LSME has no understanding of which identifiers in a programming language are reserved, and so special checks must be written to detect if a word that LSME classifies as an identifier is actually a reserved word and should be ignored.

Secondly, because it is targeted as a reverse-engineering tool, it requires more specification work to extract a useful dependency model than perhaps is necessary; a concern of Murphy and Notkin [1996] was ensuring that their notation language had the same expressiveness as the regular expression language it is based on. Because of this, LSME suffers from the same comprehensibility problems that plague regular expressions noted above, but also requires more specification work to extract a useful dependency model would be necessary if it otherwise strictly focused on dependency analysis.

Finally, LSME requires significant capability from the developer to use: not only must they correctly specify the lexical patterns needed to match dependencies, they must also provide an analysis specification to generate the dependency model could be constructed. In this ways, LSME can require more skill to use than the GREP-like tools it was intended to replace.

3.2.2 Syntactically-Aware Tools in Polylingual Contexts

Syntactically-based tools for polylingual systems are typically focused on the technologies that bridge languages for interoperability, and are configured for specific language combinations. All these approaches share some basic common strengths and weaknesses. Because the tools recognize the syntax and incorporate knowledge about some of the semantics of the languages they analyze, they are more likely to be accurate in their detection of dependencies because they are not confused by cases where there exists lexical similarity between two identifiers, yet their semantic

context indicates they are not dependent on each other. And while these approaches are cheaper to implement than semantically-aware tools, all require implementing a parser for each language analyzed: as noted before, this is often problematic in practice [Grass and Chen, 1990; Reubenstein et al., 1993; Murphy and Notkin, 1996; Devanbu, 1999].

The TAWK tool of Atkinson and Griswold [2006] (introduced in Section 3.1.2.2) does not quite fit this description. While it incorporates syntactic knowledge of the language under analysis (through a parser-generated AST representation of the developer's source code), the tool has no understanding of the language's semantics. Instead, it relies on the developer to provide regular-expression based queries, augmented with some syntactic restrictions, to perform dependency analysis. While this makes it easier to re-target TAWK for other polylingual technologies, as the semantic analysis can be rewritten, it does mean that TAWK has fewer capabilities in specifying forms of analysis than other approaches aimed squarely at syntactic analysis. At the time of publication, TAWK only supported two languages, making it likely that any application to a polylingual system would require significant upfront development costs.

Another syntactically-aware approach specifically targets the JAVA NATIVE INTERFACE (JNI) for bridging JAVA and C/C++ code, rather than attempting to solve the more general problem of polylingual dependency analysis. The work of Moise and Wong [2005] is conceptually similar to TAWK: they reuse an existing parser framework from the SOURCE NAVIGATOR IDE[11] to extract structural facts from JAVA and C/C++ source code, then apply heuristics to determine where cross-language dependencies exist. Moise and Wong [2005] observed that the contextual information needed to infer a dependency between languages was not sufficiently provided by the resulting parse-trees of the SOURCE NAVIGATOR tool, even though it had complete syntactic understanding of the languages involved; their experience mimics problems noted in the preceding decade by Reubenstein et al. [1993] and Devanbu [1999].

Inadvertently, their research highlights a problem with reusing existing parser technology. The SOURCE NAVIGATOR IDE, on which they based their technique, had not been updated for two

[11] http://sourcenav.sourceforge.net

years at the time of their work's publication, despite the fact that the JAVA language had undergone substantial revisions during the interim.[12] More recently, development of this IDE has been abandoned in favour of an exploration-only tool.[13] As a consequence, a developer looking to apply this work to their project will need to make a substantial investment in updating the outdated components of the tool before it can be successfully applied; considering that the platform is obsolete and unsupported, it is unlikely that such investment would be made.

One interesting result worth noting from their work concerns a discrepancy between how developers use the JNI bridge, and how they were supposed to use it. Moise and Wong [2005] found several cases where a dependency should exist between JAVA and C code in their case studies, but where their analysis failed. On investigation, they determined that this was because it was possible to not follow the JNI specification in the implementation, yet still have properly functioning code. For example, they discovered several cases where C code in use by JAVA code was using the standard integer primitive **int**, rather than the special joint provided by the JNI for integers being passed to JAVA code. While Moise and Wong [2005] blamed programmer inexperience for these discrepancies, it is worth noting the joint is an alias for the **int** type, created by JNI using the typeset syntax in C.[14] While JAVA 1.3 was the version used by Moise and Wong [2005], these same typesets are in use in JAVA 1.7. It is plausible that developers may be aware of this, and may decide to not use the JNI alias for int. Regardless of intention, it is worth recognizing that this actually complicates polylingual dependency analysis: configuring an analysis tool with the correct, documented syntax and even semantics of polylingual system can still miss those dependencies that are incorrect in their specification, but after processing by the polylingual technology are indistinguishable from the correct form.

[12] JAVA v5.0 was released in 2004, introducing generics and enumerations, both of which introduced new syntax: see http://www.jcp.org/en/jsr/detail?id=176.

[13] http://sourcenav.sourceforge.net

[14] http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/types.html

3.2.3 Semantically-Aware Tools in Polylingual Contexts

Semantically-aware tool support for polylingual systems does exist, often in commercial products; however, it is typically limited to supporting one specific technology and language combination. For example, the HIBERNATE Object/Relational Model framework provides the means for allowing JAVA code to interact with SQL database queries and results in an object oriented fashion, through the specification of XML-based mappings; the technology is similar to the IBATIS Object/Relational Model framework described in Section 2.1. However, while cross-referencing support is available for the HIBERNATE framework via a modelling application in the JBOSS TOOL KIT,[15] the same support does not exist for IBATIS, even though both frameworks use the same languages for the same purposes. In fact, there are at least 23 different object/relational modelling technologies for the JAVA programming language, which all solve the same problem, using the same (or some of the same) languages, but with differing semantics for describing cross-language interaction for each technology.[16] In general, when semantically-aware tool support exists for polylingual dependency analysis, it is because a particular technology has reached a critical mass where the cost of constructing tool support is commercially viable due to the size of the install base [Cossette and Walker, 2007]. While little research has attempted to identify the costs of adapting semantically-aware polylingual tool support to other languages or technologies, the lack of such support is suggestive.

There is some research on conducting dependency analysis in polylingual contexts, but these works have focused on highly specific problems. Furr and Foster [2006] created polylingual tool support for the same JNI technology that Moise and Wong [2005] did, but focused on checking the type correctness of C and JAVA interactions. Their work notes that the JNI "bridge" strips away most type information from C code, so to resolve this they use a semantic-analyzer to examine

[15] http://www.jboss.org/products/devstudio/overview/. Specifics of the modelling can be found in the documentation here: http://docs.jboss.org/tools/4.1.0.Final/en/hibernatetools/html/plugins.html#d0e2543.

[16] http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software. While I am loathe to cite Wikipedia, there is unfortunately no other resource that has attempted to list the alternatives out there. Vendors are obviously not interested in listing their competitors, and even open source projects have similar inhibitions.

the source code's interactions, infer what the correct type information should be, and then embed additional parameters and/or specially defined types to provide needed supplementary information. While only somewhat useful for dependency analysis, Furr and Foster [2006] do show that the semantic context surrounding a polylingual dependency may be missing from source code, making semantically-aware tool support that much more difficult to construct.

The work of Martin and Xie [2006] attempts to address the case of identifying JAVA methods that are dependent on the same parts of a database, by determining if the SQL queries embedded as strings in each method are similar. Their attempt is based on trying to reuse the AMNESIA tool of Halfond and Orso [2005], originally created for preventing SQL-injection attacks: AMNESIA identifies usage points of the JDBC API, and then uses a constraint-based analysis of embedded strings to determine what possible SQL queries could be constructed at that point, which are later used to monitor actual submitted queries to see if they deviate [Halfond and Orso, 2005; Halfond et al., 2006]. Martin and Xie [2006] reuse AMNESIA's ability to reconstruct SQL queries from embedded text to identify if the enclosing JAVA method is dependent on the same parts of the database as other methods with similar embedded SQL fragments.

The approach however is overly simplistic, and not much more effective than using existing tools available in any software IDE. Martin and Xie [2006] use a byte-code analysis library to find usages of the JDBC API in the compiled JAVA libraries, despite the widespread availability of semantically-aware search functionality in a modern IDE that could easily find occurrences of JDBC calls in the text of the source code. It is also unclear how necessary the construction of string automata is for discovering which embedded SQL fragments from different methods use the same table or column names. By way of contrast, a regular expression could extract the embedded fragments at those locations, which could then be refined using a basic lexical analysis that removed any known SQL reserved words, leaving only table and column identifiers behind. A basic lexical similarity measure at this point might be as effective as that of Martin and Xie [2006] in suggesting if two JAVA methods have related queries. Perhaps the limited space available at

their venue meant that Martin and Xie [2006] were not able to sufficiently elaborate on the cases in which their approach was necessary over such alternatives.

3.2.4 Partial-Language Recognition Approaches

Adapting reverse-engineering tool support for new languages or their dialects has previously been shown to be difficult, particularly when the language itself is complex (see Section 3.1.2.1 and Section 3.1.2.2); however, researchers also noted that often a subset of a language's syntax or semantics were all that was needed for specific forms of analysis [Wong et al., 1995]. Early ideas on how this could impact reverse engineering tool support can be found in the work of Bischofberger [1992], who introduced the idea of *fuzzy parsing* to reflect the notion of constructing parsers with incomplete syntactic recognition. The SNIFF tool put forward by Bischofberger [1992] was a C++ parser which implemented enough of the C++ specification to reverse-engineer useful models of the source code, and relied upon detecting . The fuzzy parsing of SNIFF provided three main advantages.

1. SNIFF was significantly faster in analyzing source code because it could ignore syntax irrelevant to the needed models.

2. The development costs of SNIFF were significantly less, as some of the more complex parts of the C++ language could be ignored.

3. The parser did not require that C++ code be correct for it to be successfully parsed, as its imprecise language recognition allowed it to ignore some syntactic errors while still extracting the structures of interest.

It is worth noting, as a reference for the costs of developing new a parser from scratch (see Section 3.1.2), that Bischofberger [1992] describes industrial examples where using SNIFF to construct C++ parsers and reverse-engineering frameworks were accomplished with a person–year's worth of effort: a cost that in their mind was a considerable savings [Bischofberger, 1992, p.15].

Unfortunately, Bischofberger [1992] does not describe *how* fuzzy parsing is implemented in SNIFF, making it difficult to build off of this work. This is addressed by Koppler [1997], who recognized that one of the key contributions of Bischofberger [1992] lay in the opportunity to construct reverse-engineering tools with sufficient semantic-awareness, but doing so cheaply by providing just enough information for one specific analysis. The work fills in the missing ideas from Bischofberger [1992] by detailing several differing approaches for implementing fuzzy parsers. The approach Koppler [1997] uses in their framework relies on the recognition of *anchor* tokens, or specific identifiers and/or symbols in the text of the source code that commonly precede an identifier of interest (e.g., the JAVA keyword **class**, which precedes the actual name of a JAVA class). Their parser ignores all input until such an anchor is detected, and then proceeds to attempt to match the subsequent tokens to one of the grammatical productions it is aware of.

Later work by Moonen [2001] improves on these ideas by using *island grammars* to construct partial language recognition parsers in a more formal manner than either Bischofberger [1992] or Koppler [1997]. Island grammars are a metaphor for describing the construction of a partial-language recognition grammar: grammatical productions are provided to recognize the key syntax of interest (the *islands*), then special *water* productions are written that are intended to capture and ignore all other possible syntax [Moonen, 2001, 2002]. While similar to how fuzzy parsers only specify key language details, island grammars are not as restricted: the island productions themselves can incorporate water productions, allowing for broader fault-tolerant recognition of syntax, and do not need to rely on identifying an anchor token before relevant syntax can be captured.

The MANGROVE tool created by Moonen [2001] as an island grammar framework was initially used to measure different software metrics (coupling, and cyclomatic complexity) in two legacy systems where each was implemented in a different language (COBOL and UNIFACE). In subsequent work, Moonen [2002] examined how island grammars could be applied for constructing dependency analysis tool support for an obscure dialect of COBOL which was no-longer supported,

and for which no other parsers were available for reuse. Both of these works are promising solutions for the problem of polylingual dependency analysis, as they show that useful tool support based on island grammars could be constructed "from scratch" for three different languages (or dialects) in a reasonable amount of time. All that is missing is the ability to combine different island grammar parsers together, and to detect dependencies across their language boundaries.

Since then, two different works have broached the idea of applying island grammars for constructing polylingual dependency analysis support. The first was a proof-of-concept paper by Synytskyy et al. [2003] which attempted to show that fault-tolerant parsing of polylingual source code was possible, by providing an island grammar specification capable of parsing one text file constructed by the authors that contained HTML, VISUAL BASIC, and JAVASCRIPT code. The example however is trivial, and no attempt is made to construct any kind of source model from the resulting parse.

The second was a systematic investigation of the usefulness of applying island grammars to polylingual dependency analysis by Cossette and Walker [2007]. Their work noted that while island grammars should be effective for this problem domain, it was not clear how difficult it was to construct reverse-engineering tool support with it, as prior applications were done by an academic researcher whose skill set was likely different than an industrial developer [Moonen, 2001, 2002]. The approach of Cossette and Walker [2007] was to first determine the polylingual dependencies present between two files in a large open source system, and then measure the accuracy of a series of island grammars in discovering those dependencies. Using their LUTHER testbed framework, the authors varied the level of syntactic and semantic awareness specified in a series of island grammars, estimated the effort needed to specify that grammar, and compared this to its dependency analysis accuracy. Their work showed that while polylingual dependency analysis was possible with island grammars, the costs of specification rose faster than the resulting grammar's accuracy, with later grammars requiring significant effort to construct. As most developers are unlikely to regularly developer grammar-based tools, or *grammarware* [Klint et al., 2005], Cossette

and Walker [2007] noted that additional simplifications for describing island grammars would be needed to limit the difficulty industrial developers would encounter in specifying them.

3.2.5 Summary: Polylingual Dependency Analysis

Polylingual software systems, where software is implemented using multiple programming languages thanks to one or more technologies that facilitate cross-language interaction, pose a difficult problem for reverse engineering tools, and dependency analysis in particular. The possible language and technology combinations that may exist in these systems makes creating general-purpose, semantically-aware tool support unrealistic. The few semantically-aware tools which are available owe their existence to their particular polylingual technology's adoption by a sufficiently large user-base that commercial development of such support is viable. Even so, these tools are tied to that particular combination of technology and languages; other similar technologies using the same programming languages cannot make use of it.

Lexically-aware and syntactically-aware approaches to dependency analysis have been, or can be, applied to polylingual systems, but neither is ideal. Lexically-aware support is the most suited for attempting to reverse-engineer a dependency model from an arbitrary combination of languages, but still relies on developer skill with specifying complex regular expressions for their effectiveness. Since these tools are largely aimed at general reverse engineering support, more language and analysis details must be specified than would be strictly necessary for dependency analysis, increasing their up-front costs to use. Syntactically-aware approaches to polylingual dependency analysis are even more impacted by the problems reverse engineering tools have in general when needing to add parsers for new languages (see Section 3.1.2). Each parser added can require significant rewrites to other existing parsers, and any analyses already in place, due to the normally tight coupling of syntax and semantics, further complicated by the polylingual technology's differences. Attempts to reuse existing parser support have shown how easy it is for them to become obsolete.

A promising approach comes from work on partial-language specification tool, involving fuzzy

parsers or island grammars, which focus on providing just enough details of a language's syntax or semantics to perform a desired analysis. The reduction in needed language syntax makes constructing support for a variety of languages from scratch feasible, and the level of semantic detail available in the partial specifications allows for dependency analysis tooling that is effective in industrial contexts. At present, the main obstacle in applying this to polylingual systems is the specification difficulty: industrial developers who do not regularly work with such grammarware would find writing an island grammar challenging, with the effort spent in providing semantically-richer island grammars rising faster than the analysis's effectiveness. What is presently needed are ways to simplify the specification of these language's details, and their polylingual interactions.

3.3 Library Evolution and API Migration

In the two previous sections, I have discussed research related to anticipating or predicting the impact of changes to software systems, by identifying entities which are dependent on the entities that are needing to be modified. Because software developers are the initiators of these changes, they can proactively investigate their planned modifications to identify their impacts, and either take steps to appropriately redress affected software entities to accommodate the change, or alter their initial modification to lessen its impact [Arnold and Bohner, 1993]. However, cases can arise where developers cannot plan such changes in advance. For serious software defects or security vulnerabilities, the necessity of immediately fixing the problem can require that developers initiate changes, and then determine how to redress any side-effects they introduce [Parnas, 1994; Kim et al., 2006; Glerum et al., 2009; Perkins et al., 2009; Giger et al., 2010; Nagappan et al., 2010; Yin et al., 2011; Zhang et al., 2013]. In other cases, the source of the change may not be in the software developer's source code, but rather in an external software entity on which they rely [Dig et al., 2006; Dig and Johnson, 2006a; Kapur et al., 2010]. Because software developers can only react to these changes, rather than plan for them, their informational needs are different: the impact of the change is often already known, thanks to a slew of compilation errors and failing unit tests [Kapur

et al., 2010]. Instead, software developers need help identifying how to redress software entities that were dependent on the functionality that was altered.

Change impact stemming from software library's and their APIs is a particularly difficult aspect of this problem. Software libraries are collections of software entities that provide useful functionality for other software developers to reuse in constructing software, and is one aspect of the broader area of software reuse [McIlroy, 1968; Poulin et al., 1993; Johnson, 1997; van Gurp and Bosch, 2001; Holmes, 2008]. These libraries provide an API which serves as an abstraction later to separate the provided functionality and usage details from the underlying implementation [Parnas and Siewiorek, 1975; Pancake, 1995; Kramer, 2007]. Client programs use functionality provided by the API by directly invoking functionality, or by building on top of it through specialization points to customize it for their needs.

The role of the API in insulating client programs from changes to the library's implementation is grows in importance as the library's adoption becomes widespread. Any change to the library's implementation that necessitates a change in the API impacts every software system that uses it: thus, software library developers are encouraged to treat APIs as contracts that never change once publicly released [Bloch, 2006; Jacobson et al., 2011]. An extreme example of this is the recommendation that library developers never provide a version number for their APIs, thus indicating it will not change and instilling confidence in its users that no future changes will harm their dependent software [Jacobson et al., 2011, Chapter 5].[17]

This idea that the API can never change, and will insulate client software from any necessary changes to the library, echoes earlier themes in change impact that proper upfront design can contain future changes (see Section 3.1). However, software libraries are subject to the evolutionary pressures that all software systems face, and some future changes deemed necessary will

[17]Jacobson et al. [2011] give as an example of this practice the National Public Radio (NPR) API which was versionless during its creation, and subsequent release in 2008. In praising this practice, they say that "[a]ll apps built on top of this API enjoy great stability and confidence that the API will continue to evolve without harming their apps" [Jacobson et al., 2011, Chapter 5]. Since then, the NPR API has become a counterexample to this argument: it has now adopted versioning (v0.93 is the current release) and has also altered the API, according to http://www.npr.org/api/index, as of June 19, 2014.

require changing the API during their enactment [Xing and Stroulia, 2007b; Dagenais and Robillard, 2011]. As David Ungar, then a "Distinguished Engineer" with Sun Microsystems, put it:

> "... the notion of interface is much more of an illusion than people give it credit for...You try to get all your design decisions to hide behind interfaces so that changes percolate through as few levels as possible. But every honest programmer will admit there are times when you have to change your mind, and the interface changes" [Pancake, 1995, p.35].

API evolution is often necessary, but causes problems for both the library's developers, and the client developers whose software relies on it. For client developer, changes in a new library version's API that break backwards-compatibility with the prior releases API represent a major inconvenience that requires unplanned modifications to their source code to accommodate [Perkins, 2005]. For the library developer, the decision to alter the API is not easy, as breaking changes can alienate the library consumers, causing client software developers to choose to continue to use older library versions, or look for alternative libraries to provide the same functionality [Bloch, 2006; Jacobson et al., 2011]. The question is whether the impact of these changes can be avoided, or if not, whether developers can be assisted in their redress.

3.3.1 Proactive Library Migration

It is not always the case that the only option is to react to such changes; several avenues of research have been proposed to address change impact stemming from software libraries through proactive measures. Such approaches rely on the library developer manually (or through tool support) documenting how their library's API has changed, and how dependent code should be migrated.

Chow and Notkin [1996] provide a semi-automated library migration tool that can be used to transform a software system's usage of an API to its latest version, by enacting the necessary adaptations in the source code for the client developer. Their approach relies on the library devel-

oper providing a formal change specification for the new API version including how the API has changed, and the code transformations necessary to migrate dependent code to accommodate the change. From this, a source code transformation tool can then be generated to identify and migrate references to outdated portions of the API in a software system. This can be applied by client developers at their discretion on their software, choosing how and when to migrate their code to the next library version.

A fully automated approach similar to Chow and Notkin [1996] is provided by Henkel and Diwan [2005] in their CATCHUP! Eclipse plugin. CATCHUP! targets a subset of API changes, focusing on supporting the migration of behaviour-preserving, structural transformations (called *refactorings*) to the API. While Chow and Notkin [1996] require the library developer to manually specify transformations, CATCHUP! has library developers mark when they use the automated refactoring tool support in the Eclipse IDE to perform an API change. CATCHUP! records each refactoring, and saves the sequence of API refactorings into a migration script that can later be given to client developers to migrate to the new version of the library. This script allows CATCHUP! to re-enact all of these refactorings on the client developer's source code, thus transforming it to use the new API version. While Henkel and Diwan [2005] restrict the supported API changes in CATCHUP! to refactorings, in so doing they free the library developer from having to do much additional work to specify the needed migrations. It is worth noting this has become part of the standard tool support available in the Eclipse IDE.[18]

The REBA Eclipse plugin of Dig et al. [2008] works in a similar fashion: refactorings that library developers perform on their API are tracked in the Eclipse IDE, and recorded into a script. REBA then uses this script to generate an additional abstraction layer that mimics the original API, effectively using the sequence of refactorings to generate adaptive code that transform usages of the old API into usages of the new. Client developers can then use the new library without modify their software's source code, which is advantageous for legacy software systems where the risks associated with code modification are often high [Parnas, 1994].

[18] Since Eclipse v3.2.

If a library developer is proactive in supporting their API changes, the above approaches can offer a significant effort savings to client developers reacting to library evolution. However, each of the above approaches limits the kinds of API changes that can be supported. Both CATCHUP! and REBA only support API changes for which Eclipse has automated refactoring tool support, which already is a subset of possible refactorings. While Chow and Notkin [1996] have different restrictions on the supported transformations, it requires additional effort from the library developer in manually writing transformations for each change. The approach also offers additional risk if these transformations are incorrect: while CATCHUP! and REBA can rely on Eclipse's refactoring engine to ensure the correctness of the transformations, Chow and Notkin [1996] has no such facility, and must rely on the library developer's skill in correctly identifying and specifying the needed transformations. The effectiveness of these approaches then rests on whether: (i) the library developer is proactive; (ii) the API changes fall within the categories of supported transformations; and, (iii) the library developer is able to correctly and completely specify the transformations, in some cases without the assistance of tool support.

3.3.2 Reactive Library Migration

In the absence of guidance or assistance from library developers, client developers are on their own to discover how to migrate their code in response to API change [Perkins, 2005; Xing and Stroulia, 2007b; Kapur et al., 2010; Nita and Notkin, 2010; Wu et al., 2010; Dagenais and Robillard, 2011]. This is not an easy problem: while it is clear what is missing from the new API, information about what caused the change, and what the alternative functionality should be, is missing in this case. Client developers need to discover if replacement functionality exists, before they can proceed to actually redressing the impact on their software.

A starting point in addressing this problem is to try and find where missing API functionality went to, by performing *origin analysis* [Godfrey and Zou, 2005]. Origin analysis is the idea of reconstructing the evolutionary history of a software system to trace the relationship between new additions and modifications/removals: essentially, it seeks to answer the questions "Where did

this go?", or "Where did this come from?". By analyzing the implementations of the altered functionality, origin analysis techniques seek to determine if what appears to be a removal of functionality from one version of a software system (e.g., a library's API), is instead a movement of its functionality into an entity elsewhere in that version.

The BEAGLE tool by Godfrey and Zou [2005] is an early example of an origin analysis tool for detecting the merging and splitting of functionality between subsequent versions of a software system. BEAGLE uses a series of analyses based on lexical similarity, software metrics, and a static analysis of dependency relationships (see Section 3.1.2), to determine if new or removed functionality in fact represents the moving, splitting, or merging of functionality from another entity. BEAGLE is a *semi-automatic* approach to origin analysis: despite performing these analyses, BEAGLE does not determine the origin of functionality on its own, but presents the possible origins for missing functionality along with the ratings its analysis suggests, and asks a developer to make the determination as to which origin is correct.

Work subsequent to Godfrey and Zou [2005] focused on determining how origin analysis could move from semi-automated analyses to fully-automated. An early attempt by Kim et al. [2005] added additional analyses to those used by BEAGLE, and then automated the analysis by combining them into a weighted similarity measure, and selecting the most similar entity as the origin of missing functionality. The core assumption in this work is that the battery of metrics used, combined with the weighting system, is enough to accurately resolve all questions relating to origin analysis. A variant on this is the UMLDIFF tool by Xing and Stroulia [2005], which does not directly analyze the source code, but rather a reverse-engineered model of the code based on the Unified Modelling Language (UML)[19]. UMLDIFF supports a smaller taxonomy of possible changes recovered through origin analysis: primarily, it detects the movement and renaming of functionality, and requires that the majority of the software has not changed so as to simplify the difficulty of analysis.

A different approach for automating origin analysis is to restrict analysis to those cases that are

[19] http://www.uml.org/

easily verifiable (similar to restricting library migration support to refactoring-based changes, as in Section 3.3.1). Weißgerber and Diehl [2006] also performs an automated origin analysis using the change sets tracked by the software's version control system (VCS), with the goal of identifying the origin of functionality, and the actual code transformation. To do this, Weißgerber and Diehl [2006] restricted their analysis to recovering transformations resulting from refactorings between two versions of software, by using a lightweight lexical parser (see Section 3.1.2.2) to identify classes, interfaces, methods, and fields that changed, and then applying a series of heuristics aimed at identify which (if any) refactoring best explained that change. To eliminate accidental matches to non-refactored code, they tried to determine if the implementation of refactored classes and methods was largely unchanged using *code clone* detection techniques. If the implementations were largely identical, or clones of each other, it suggests the behaviour of the functionality was preserved despite the structural changes made—a requisite for most refactorings.[20]

Kim et al. [2007] blends many of the above approaches in automating their origin analysis technique: like Godfrey and Zou [2005] and Kim et al. [2005], they rely on lexical similarity to identify possible origins for functionality between software versions; but like Weißgerber and Diehl [2006], they apply a series of heuristics aimed at recovering what transformation could best explain their change. Their goal is to recover broad change rules that collectively describe all these differences: for example, a series of refactorings that changed return types on different methods in a consistent fashion could be represented as one change rule. The approach comes with a drawback: the resulting change rules are superficial in their descriptive power. The possible transformations considered in identifying change rules are largely limited to those causing identifier renames,[21] but because the analysis is lexical, the checked transformations do not distinguish between trivial or complex reasons requiring that change. For example, the *typeReplace* rule change does not distinguish between a simple class rename refactoring that caused references to change, or a more serious reworking of functionality resulting in a drastic type change, and thus widespread side-

[20] But not necessarily all [Opdyke, 1992].

[21] With some special cases for multiple identifier renames in parameter lists, and covering the addition or deletion of parameters from method declarations

effects to the system. Thus, the explanatory power of the change rule provided by Kim et al. [2007] may be broader, but not as deep; Weißgerber and Diehl [2006] by comparison attempt to make such distinctions, but would not report a serious reworking of functionality as it would not be a refactoring.

3.3.3 Recommending appropriate reactions

Origin analysis tools are not always appropriate to use in resolving issues stemming from library migration. When functionality is removed from an API in a new library release, applying an origin analysis tool will attempt to discover if this functionality still exists in the library, but has been renamed or merged into some other library entity. But discovering what happened to missing functionality is not the same as discovering what to use in its place. Consider the case where a library developer decides that functionality in the API should be for internal use only, and removes it by reducing its visibility (e.g., by making it **private** in JAVA): origin analysis will discover the same (now hidden) method; however, what the client developer wants to know is what API functionality they should be using instead. Tools that can recommend replacement functionality for these cases are referred to as *API change recommenders*.

The REFACTORINGCRAWLER tool of Dig et al. [2006] shows the beginning of this movement away from origin analysis towards API change recommendation. Motivated by empirical research suggesting most API changes were caused by refactorings [Dig and Johnson, 2006a], REFACTORINGCRAWLER attempted to identify the refactorings which may have occurred to the public API of a software library between versions. At a conceptual level, REFACTORINGCRAWLER's approach is similar to that of Weißgerber and Diehl [2006]: both techniques initially determine possible points of change between two versions of a software system, and then apply a transformation analysis to discover what refactorings could explain those changes. The primary differences lay in their analyses:

- REFACTORINGCRAWLER uses a syntactic parser to generate an AST representa-

tion of both versions of the source code, rather than using regular expressions to extract changed identifiers, providing more syntactic information about the underlying source code;

- Weißgerber and Diehl attempts to find possible origin candidates between each version by examining the similarity of their declarations, while REFACTORING-CRAWLER looks for similarity between their implementations; and,

- Weißgerber and Diehl uses a lexical examination of the changes to the declarations to determine which refactoring was applied, using code-clone detectors to prevent non-refactoring matches, while REFACTORINGCRAWLER uses semantic analysis on the software system's implementation to see if the detected refactorings are logically consistent with related changes in the source code.

Unlike general origin analysis tools, Dig et al. restricted their tool to work with public entities in a software library, specifically to avoid detecting changes to internal components that were not part of the API, and thus not of interest to client developers. This reflects the intended goal of allowing the application of proactive library migration techniques (see Section 3.3.1) in cases where library's developers had not pre-identified the adaptations necessary to redress their changes. But, in many respects REFACTORINGCRAWLER is still an origin analysis tool: the *PushDownMethod* transformation is the only case detected in which functionality can be removed from the API without replacement, and REFACTORINGCRAWLER cannot recommend alternatives in this case.

The work of Xing and Stroulia [2007b] was one of the first to recommend alternative functionality when entities were removed from an API without replacement. As with prior approaches, their DIFF-CATCHUP tool first identifies points of change between two versions of a library (using their prior UMLDIFF tool [Xing and Stroulia, 2007a]), and then applies analyses to determine the most plausible explanation for the change. Where Xing and Stroulia differ from previous approaches is in recognizing that identifying the library change does not necessarily lead to the identification of replacement functionality: in cases where their UMLDIFF is unable to match API functionality,

suggesting that it has been removed from the API without replacement, DIFF-CATCHUP applies heuristics to identify other entities within the API that may offer an alternative. The heuristics are influenced by the structural and architectural relationships represented in UML class diagrams and (aside from lexical name similarity in the identifier) focus on the inheritance, usage, and type equivalence between the declarations of missing and new API entities to suggest related functionality. Like the BEAGLE tool (see Section 3.3.2), DIFF-CATCHUP is a semi-automatic approach in that it does not automatically determine replacement functionality in these cases, but provides the developer with a list of candidate replacements, along with a justification for their selection.

The origin analysis and API change recommender approaches previously discussed have all relied on heuristics to suggest how functionality has moved, or what its replacement should be. Because these are approximate solutions, other forms of meta-analysis to determine how to chose between rival candidates, ranging from requiring the developer to determine the correct replacement [Godfrey and Zou, 2005; Xing and Stroulia, 2007b], empirically deriving a weighted similarity measure [Kim et al., 2005, 2007], or restricting changes considered to those whose analysis is simpler (such as refactorings [Xing and Stroulia, 2005; Weißgerber and Diehl, 2006; Dig et al., 2006]).

A different paradigm was put forward by Dagenais and Robillard [2011] with their SEMDIFF tool, which introduced a new analysis technique that I refer to as *call analysis*. SEMDIFF is based on the recognition that in some cases, the underlying implementation of a library may make use of its own API, and may itself be affected by API changes in a new release. If so, examining the differences between the library's usage of its API between releases can suggest replacement functionality. Like Weißgerber and Diehl [2006] (see Section 3.3.2), SEMDIFF operates on the revision history of a system as stored in a VCS, examining incremental changes made to the source code of the library, looking for calls to the library's API (or, *callers*), and calls from the library's API to the library's implementation (or, *callees*). What SEMDIFF looks for is how the library migrates its own functionality in response to the changes in the API: when an API entity is removed, a developer

can use the tool to find how the library altered its own references to that API as a suggestion to how they can migrate their own code. Because it is possible that multiple, different API elements may be suitable replacements depending on the calling context, SEMDIFF offers support and confidence measurements for each replacement, indicating the frequency an adaptation was used to repair a specific usage of the API, and how many other adaptations were elsewhere used to repair the same. A variation on this approach is that of Schäfer et al. [2008] which applies a simpler form of call analysis through association-rule mining on two versions of a library, rather than a per-change analysis of a system's revision history. Schäfer et al. [2008] also introduces analyzing the example or test code which may accompany the library, as well as any other systems which have already been migrated to the new API, for additional examples of adaptations to the API changes.

While the underlying call analysis technique is very useful, both Dagenais and Robillard [2011], and Schäfer et al. [2008], found it necessary to provide means to expand the pool of examples that call analysis could be applied to. In part, this is because the success of this technique relies on the degree to which a library is self-referential: if functionality is removed which the library did not make use of, call analysis is of no use.

The work of [Wu et al., 2010] attempts to improve on this limitation in their AURA tool, which *hybridizes* the call analysis approach of Schäfer et al. [2008] with the origin analysis work of Kim et al. [2007]. AURA starts by applying call analysis to detect replacement functionality, but then uses the lexical and syntactic analyses of Kim et al. [2007] for those cases where the library does not use that API functionality, broadening its applicability. Like Kim et al., AURA tries to analyze all changes to a library's implementation, and thus may identify changes to non-API functionality which are not of interest to client developers. Similarly, the LIBSYNC tool of Nguyen et al. [2010] attempts to expand the capability of between-version call analysis by incorporating additional analyses on already-migrated programs. They use a clone-detector on the JAVA bytecode representation of each library version to detect simple move and rename operations, similar to DIFF-CATCHUP's use of the UMLDIFF tool [Xing and Stroulia, 2007b]. Their application of call

analysis is similar to Schäfer et al. in that it is focused on gleaning usage examples from programs that have already migrated to the new library version, by comparing their implementations between version. However, they augment this by using *program slicing* [Weiser, 1984] to identify other lines of code (presumably just in the method) that are data or control-flow dependent on the line of code identified by call analysis as a usage change. Like Schäfer et al., LIBSYNC does not distinguish between usage changes that redress the impact of changes to the library's API on the program, and usage changes that are unrelated to breaking API changes.

Some change recommenders employ a "best-of-breed" paradigm in their hybridization, such as the HIMA tool of Meng et al. [2012]. HIMA combines four prior approaches in a five-step sequence[22] to detect changes between two library versions:

1. parsing check-in comments from version control systems for the associated file modifications, identifying keywords that might suggesting the underlying change (as in Weißgerber and Diehl [2006]);

2. applying call analysis to the incremental changes made over the library's development (as in [Dagenais and Robillard, 2011])[23];

3. applying heuristics to detect if modifications match known refactorings (as in Weißgerber and Diehl [2006]);

4. marking unresolved changes as API deletions (as in Wu et al. [2010]); and,

5. aggregating identified change rules into broad, generic rules with wider applicability (as in Kim et al. [2007]).

[22] Similar to the "chain-of-responsibility" design pattern [Gamma et al., 1994]

[23] It is worth noting that while Meng et al. [2012] discuss applying call analysis to the incremental changes made to the library's implementation, there is no mention of how they resolve the problem of ambiguity present in small code fragments. The lack of contextual information (e.g., import statements, prior type declarations, etc.) in the committed change make it impossible to precisely determine the entities involved. Both Kapur et al. [2010] and Dagenais and Robillard [2011] use partial program analysis [Dagenais and Hendren, 2008] to resolve such ambiguities; however, Meng et al. does not indicate how they deal with this.

Meng et al. do not indicate if the sequence of applied techniques is significant, or if other approaches were found ineffective in developing the HIMA tool, and relies on hybridized techniques having high precision in their analysis to successfully aggregate their results. HIMA cannot incorporate low or moderate precision analysis techniques that may be useful in specific contexts.

3.3.4 Redressing library change impact

Once a developer has determined the set of adaptations needed to migrate their code to a library's new API, the problem remains of enacting those changes. While developers can manually enact changes by editing code, their systematic and repetitive nature increases the risk of human error during their modification [Kapur et al., 2010; Holmes and Walker, 2012]. While several automatic migrations were discussed earlier (see Section 3.3.1), they relied on a library developer providing change specifications for the migration in advance: they do not consider cases where the client developer is specifying transformations they discovered, or needing to alter the migrations to reflect peculiarities of their software systems.

An initial, limited approach by Perkins [2005] was to *inline* the removed API's functionality into the developer's source code: that is, copy the actual implementation of the old API into all those locations in the developer's source code which called for it. Obviously this is not always appropriate: only simple functionality could be successfully inlined in such a manner, and if the affected API entities were removed due to defects or security vulnerabilities, these would be copied into the developer's source code. Further, this does not actually migrate the source code to use the new library, but effectively severs those dependent portions of the source code from ever reusing the library's functionality.

Nita and Notkin [2010] updated earlier work by Chow and Notkin (see Section 3.3.1), by supporting the provision of change specifications by client developer describing differences between two API versions. This is used to generate a customized version of their software which is transformed to use the new API, through a process called *twinning*. A core feature of this approach is that the client developer's underlying source code is not affected: the specification's mapping is

used to copy and transform the base software into a version which contains the needed adaptations, allowing the base software to be maintained separately from the migration specification.

Nita and Notkin requires a complex specification of the mapping relationship between two APIs to be of use. A simpler approach, focusing on migrating developer code in response to API changes, is that of Balaban et al. [2005] which allows client developers to specify refactorings on declarations they do not "own". In their work, Balaban et al. address the problem where a library the client developer reuses has provided replacement functionality for entities which are either *deprecated* and intended for eventual removal, or cannot have the newer capabilities (e.g., synchronization support) that other entities introduce. However, the refactoring tools do not work in these cases, as they are predicated on transforming the *declarations* of entities and updating dependent references: here, the client must initiate changes to references. Balaban et al. provides support for the developer to specify these mappings in a simple manner, and uses a type-constraint analysis to identify affected references, and migrate them correctly. However, Balaban et al. is concerned with the case where the developer has access to both the old and new functionality at the same time, and whose software is correctly compiling.

The TRIDENT tool of Kapur et al. [2010] focuses on the more common occurrence of library migration: a client developer incorporates a new library version into their software, notices all of the compilation errors, and sets about addressing each individually. In this case, the approach of Balaban et al. [2005] does not work, as the client developer is operating on references whose declarations are missing, or *dangling references*. Such dangling references lack the semantic information Balaban et al. needs for their type-constraint analysis. TRIDENT provides instead semi-automatic, refactoring-like support by allowing client-developers to select a dangling reference, and match the reference to other related references to the same deleted functionality, by using heuristic, constraint-based reasoning as to what the missing declaration is. The developer can then specify an exemplar of how these references should be transformed to use the new API, which TRIDENT then enacts on the specified dangling references. Like Balaban et al., Kapur et al. fo-

cuses on matching normal developer usage patterns with the tool's design, including the ability to preview migration enactments, and reverting mistaken transformations.

3.3.5 Summary: Library Migration and API Evolution

Software libraries offer functionality that can be reused by developers when constructing their software. These libraries offer an application programming interface (API) to access their functionality, which is intended to insulate software using the API from changes to the library's implementation. However, libraries are subject to the evolutionary pressures all software systems face, and some changes also require their APIs to be altered in non-backwards compatible ways. As new versions of the library and its API are released, migrating software to use their new versions requires redressing the impact of their API changes.

While some proactive approaches for addressing library evolution exist, these all rely on the library's developers performing additional work by providing specifications on how their APIs have changed, and how they should be redressed; further, these approaches only support a restricted set of possible API changes. But as many libraries are released without such specifications or assistance, techniques to support client developers in discovering the necessary adaptations. Approaches exist for assisting client developers in semi-automatically migrating their software to redress the impact of API changes, but these require that the client developer has previously discovered the necessary adaptations for each change first.

Early approaches for redressing API evolution focused on finding if functionality had moved in a new library version, referred to as origin analysis. Later work attempted to recover the nature of the change involved, which could suggest the needed adaptation as well. But discovering where functionality has moved to does not identify replacement functionality if it is also removed from the API. While the library developer may provide alternatives in such cases, the client developer must often discover these on their own.

API change recommenders seek to address this problem by recommending adaptations for API changes, even in cases where the original functionality is no longer available. Initial techniques

focused on recovering refactoring-based changes to the API, but later work recognized that this did not address all changes, and attempted to suggest alternatives for cases where functionality was removed. These approaches largely rely on the lexical, syntactic, or semantic similarity of the missing entity's declaration with other API entities to suggest equivalence. They can be confused by cases where significant alterations were made to their declarations, even though their implementation is largely intact.

More recent work has focused on recovering usage examples of the API to recommend as adaptations through call analysis, which examines how calls-to, and calls-from the library's usage of its own were redressed in subsequent versions. While capable of recovering more sophisticated API changes than prior work, it was recognized early that such techniques are limited by the degree to which the library used its API in its implementation. Subsequent approaches have hybridized call analysis with other origin analysis techniques to expand their coverage of API changes.

3.4 Empirical Studies of API Change

In Section 3.3, a variety of approaches were presented for discovering how the APIs of software libraries have changed between versions, and to suggest how dependent code in software systems that are clients of the library should be redressed. The approaches thus far have a common problem: their evaluations are limited in identifying how well they work, as none have been tested on all API changes in a library. Researchers need data describing how APIs have evolved so as to interpret whether a technique's output is correct, but there are few empirical studies on how software library APIs evolve, and to-date there is no corpus of software library change data that is publicly available for tool verification and testing.

As a consequence, nearly all of the evaluations of API change recommendation tools described in Section 3.3 utilize no *a priori* knowledge of what the actual changes and replacements should be, when evaluating their approaches. The most significant consequence of this is that we currently do not understand where these approaches do not work, and thus lack understanding of where

improvements are needed.

3.4.1 Empirically Determining How Libraries Have Changed

There are few empirical studies on how APIs evolve. Finding actual points of change in libraries is painstaking work [Cossette and Walker, 2007], and determining how undocumented changes to an API should be addressed requires substantial investigation.

The most prominent empirical work in this area is the case study of Dig and Johnson [2006b] on the evolution of five different JAVA APIs: Eclipse, APACHE STRUTS,[24] APACHE LOG4J,[25] JHOTDRAW,[26] and a mortgage framework from an undisclosed company. For each library, framework, or application, they chose two subsequent versions, and tried to identify all of the changes which had been made to their API using up to four methodologies:

1. examining release notes, change logs, and other developer provided documentation for indications as to which API entities were being altered, and what their replacement functionality should be;

2. interviewing the developers of an API for their recollection of changes;

3. manual investigation of the source code; and,

4. heuristics aimed at identifying possible occurrences of refactorings in the code.

For each migration, Dig and Johnson determined how many breaking and non-breaking changes occurred, and then classified each change as a refactorable change or non-refactorable. They also provided an additional breakdown as to the kinds of changes they encountered, classifying 20 different refactorings, and 4 non-refactoring changes which occurred across the systems.

Dig and Johnson found that the vast majority of changes were caused by refactorings to API entities. Their conclusion states:

[24] http://struts.apache.org/
[25] http://logging.apache.org/log4j/1.2/
[26] http://www.jhotdraw.org/

> "We found out that between the two versions we studied for each of the five systems, more than 80% of the breaking changes were refactorings. If all of the breaking changes were refactorings, it would make sense to use a refactoring tool to incorporate these changes in the applications. However, we feel that if tools could handle even 80% of the changes, they would reduce the burden of manual upgrades" [Dig and Johnson, 2006b, pp. 103–104].

This is actually an undersell of their findings: for the five APIs examined, the percentages of changes they attributed to being caused by refactorings was 81%, 84%, 90%, 94%, and 97%; 80% is the lowest proportion of refactoring-based API changes they found. The implications, as made clear in the quote, are strong: if at least 80% of API changes are caused by refactorings, then the refactoring-based approaches of Henkel and Diwan [2005]; Balaban et al. [2005]; Weißgerber and Diehl [2006]; Dig et al. [2006, 2008] and Kapur et al. [2010] would suffice to address a large majority of the work of library migration. While the remaining 20% may represent difficult adaptations to discover and enact, the client developer's workload would be drastically reduced for any library migration.

The work of Dig and Johnson is important both for its implications, and by its stature as the first empirical study of this kind. Nevertheless, their work has several shortcomings.

1. Their work analyzes a single transition between each library: while this represents significant effort, it limits the generalizability of these results.

2. The analysis relies primarily on developer-provided documentation of the changes, supplemented with additional analysis and manual investigation of the code. It is difficult to determine how complete their list of changes is.

3. Their study treats deprecated entities in the APIs as if they were a breaking change, and analyzes them accordingly: their reasoning is that deprecation implies the en-

tity will be removed soon. But deprecated entities are sometimes never removed from an API, while others are removed without being deprecated first [Kapur et al., 2010]. It is not clear that these are representative of actual changes to occur, or would have the same characteristics of non-deprecated changes.

4. The study restricts their analysis to those changes affecting entities marked as **public** in JAVA. This excludes from consideration all entities marked as **protected** from the analysis, which are typically used to provide API specialization points through subclassing and extension in object-oriented frameworks [Johnson, 1997].

5. The analysis was also restricted to those entities in the API which were "intended for reuse" [Dig and Johnson, 2006b, pp. 85, 89]. In their work, Dig and Johnson [2006b] make a distinction between entities that are **public**, part of the API, and intended to be reused, which they refer to as "published", and those which are **public**, part of the API, but whose naming suggests they are meant for internal use. This ignores supporting developers who use the APIs in unexpected ways.

Subsequent work by Kapur et al. [2010] examined the evolution of three libraries (HTMLUNIT[27], JDOM[28], and APACHE LOG4J) across a total of 54 versions, by studying the introduction of *binary incompatible* changes to their APIs. In JAVA, binary compatibility between library versions occurs when software dependent on the older version of the library can use the new library version without needing to recompile or relink the code: a binary incompatible change to the API breaks this property.

Their work reported on the number of binary incompatibilities introduced at each subsequent version of the library, finding that these libraries changed significantly throughout their lifetimes. Kapur et al. also found that the notion of APIs becoming mature stabilizing over time did not hold in all cases, as one library they examined had more binary incompatibilities introduced after its

[27] http://htmlunit.sourceforge.net/
[28] http://jdom.org/

2.0 release than at any point prior. Kapur et al. also provided some limited observational data on the role of deprecation in these libraries, noting that many changes were never deprecated prior to becoming a binary incompatible change, while other deprecated entities were never removed. However, the results of the study were limited and coarse-grained: no attempt was made to classify or resolve any of binary incompatible changes found.

The only other result in this area is an "estimate" by Henkel and Diwan [2005] that 59% of the deprecated methods and class constructors in the JAVA 1.4.2 SDK could be replaced with refactorings. They provide no details of how they arrived at this figure.

3.4.2 Evaluating in the Absence of *A Priori* Knowledge

With one exception, all of the approaches presented in Section 3.3 have performed some form of evaluation as to their effectiveness. Only Dig et al. [2006] (when evaluating REFACTORING-CRAWLER) identified library change data independently from, and prior to the evaluation of their tool. Aside from this, other researchers have attempted various methodologies to cope with the lack of publicly available library change data, such as:

- Searching for keywords like "refactoring" in the comments associated with committed revisions in a version control system, and examining files associated with those comments for relevant change examples [Weißgerber and Diehl, 2006]. The assumption of course is that all changes which are relevant will have such keywords present in their commit comment.

- Restricting the evaluation of the tool to only a few API changes that are either well documented by the library developer, or were investigated by a panel of developers to determine their causes and replacements [Balaban et al., 2005; Kim et al., 2005; Perkins, 2005; Nguyen et al., 2010]. In so doing, researchers cannot say how their approach generalizes to the other changes in the same API.

- Using a research prototype to create change recommendations for a given library,

and then manually investigating each, or a random subset of its recommendations for correctness [Kim et al., 2007; Xing and Stroulia, 2007b; Schäfer et al., 2008; Nguyen et al., 2010; Wu et al., 2010; Meng et al., 2012]. Researchers will have no idea as to what changes their tools miss, or if they are biased into believing their tool's results.

- Using existing origin analysis or API change recommender tools to analyze a library, and compare their results to the research prototype to contrast its effectiveness [Kim et al., 2007; Schäfer et al., 2008; Nguyen et al., 2010; Wu et al., 2010; Meng et al., 2012]. The evaluation can only say in which ways it is better or worse than the other tool.

- Having developers participate in an experiment where they enact a change to a library's API, and then see if the research tool can redress it [Dig et al., 2008].

- Choosing a sample of negative recommendations to evaluate the recall of the technique. In the work of Weißgerber and Diehl [2006], the sample represented 0.55% of the entire dataset.

- Relaxing weights or thresholding values so that they capture more spurious results, and investigating to determine if they contain correct change recommendations [Xing and Stroulia, 2005].

- Providing examples of cases in which the technique works [Andersen and Lawall, 2010].

- Providing no evaluation at all of the technique's effectiveness [Chow and Notkin, 1996; Henkel and Diwan, 2005].

Other researchers have chosen to evaluate their tools and techniques through case studies, in which a library migration is enacted using a prototype tool [Xing and Stroulia, 2007b; Dig et al., 2008;

Kapur et al., 2010; Nita and Notkin, 2010; Dagenais and Robillard, 2011], or a few specific changes in several software systems are migrated using a tool [Balaban et al., 2005].Performing actual migration tasks provides a fair evaluation of whether the tool and technique deals with all the necessary cases, as the compiler and/or the run-time behaviour of the program will catch missed ones; the drawback is that the changes these tools are evaluated on tend to be a small subset of the potential changes that exist in that library, or in software APIs in general. There is no reason, without digging deeper, to presuppose that these are representative scenarios.

3.4.3 Summary: Empirical Studies of API Change

While there are a variety of techniques available to assist developers in redressing the impact of changes stemming from library and API evolution, there are no complete sets of library change data to evaluate these techniques on. Consequently, researchers have not had *a priori* knowledge of the changes made to the libraries under analysis, limiting the evaluations of their work in three ways:

- researchers may not spend the time necessary to thoroughly investigate a tool recommendation's correctness, potentially biasing their assessment towards the tool;
- detailed evaluations must come at expense of generalizing to other changes in the same API; and,
- researchers cannot assess what recommendations a tool failed to make.

The most significant limitation is the latter: the lack of publicly available, complete library change data has resulted in evaluations that cannot inform us those cases for which a given technique fails. Knowing the limitations of current approaches is crucial to understanding how to direct future research.

A contributor to this problem is the lack of empirical research on how software libraries and their APIs evolve. There are only two studies of note, and while contributing to our understanding

of API evolution, each has significant limitations: one analyzed only a few migrations, and did not capture all breaking API changes, while the other measured quantitative counts of breaking changes. Understanding how software libraries evolve, grounded in the complete details of how an API changed and why, will be necessary for discovering if existing approaches are limited, and where future research should be directed.

3.5 Summary: Related Work

Proactive approaches towards addressing change impact have found that prevention through anticipatory design does not always work, and identifying the side-effects of changes before they occur is necessary to properly mitigate their impact. Since accurately detecting the impact of changes is an undecidable problem, techniques to approximately determine the dependency relationships were developed, based on reverse-engineering models of the source code. A common application was to cross-reference entity declarations against their usages, allowing developers to see locations affected by a change to that entity.

However, building such automated tool support with semantic understanding of a language is expensive, and not easily adapted to new languages or dialects. This problem is amplified in polylingual software systems, where the implementation can consist of multiple languages, and multiple cross-language facilitating technologies, each of which requires specialized dependency analysis support. But only some of a language's syntax and semantics are necessary for understanding dependency relationships; if client-developers could easily provide partial-language specifications, dependency analysis support could be constructed for polylingual contexts in a semi-automated fashion.

Reactive approaches for addressing change impact are the subject of ongoing research in software library evolution and API migration. Here, the library's developer can initiate changes to a new version of the library, impacting any software system that was dependent on the library for reusable functionality. The client developer using the library in their software must react to these

changes when they migrate to the new version, by determining how to redress their impact to the software. Two different kinds of support have been proffered for API migration redress in both fully-automated, and semi-automated fashion: origin analysis, which attempts to determine if an API change actually represents the movement of functionality, and suggest is its new location for use instead; and, API change recommenders that search for equivalent functionality in the API to serve as a substitute for removed entities.

However, our present understanding of the effectiveness of these approaches is limited, as there are no complete sets of library change data to evaluate these techniques on. Without *a priori* knowledge of all library changes, evaluations are limited in identifying their accuracy, and particularly cannot speak to those cases where the techniques fail to identify relevant API functionality for redress. A few empirical studies offer limited insight in this area, suggesting the majority of these API changes are caused by easily redressed refactorings, but it is not clear how pervasive this result is. A clear picture of how software libraries and their APIs evolve, resulting in a ground truth set of API migrations, will be necessary prior to determine future directions of research on redressing library migration.

Chapter 4

Supporting Proactive Polylingual Change Impact Analysis

The growth of polylingual software systems as a development methodology reflects the desire of developers to use languages most suited to the problem at hand [Burson et al., 1990; Hunt and Thomas, 1999; Cossette and Walker, 2007]. As with the example in Section 2.1, polylingual development is facilitated by technologies for inter-language communication and interaction, using specific dependency semantics that are often extraneous to the involved language's syntax [Moise and Wong, 2005; Furr and Foster, 2006; Kontogiannis et al., 2006; Cossette and Walker, 2007]. This poses problems for syntactically and semantically-aware approaches to dependency analysis, as the effort necessary for developing automated tool support capable of handling all language and technology combinations is enormous [Reubenstein et al., 1993; Biggerstaff et al., 1994].

However, the key syntax and semantics necessary for recognizing a dependency relationship is often a small subset of a language's specification [Wong et al., 1995], and this is also true in polylingual systems [Moonen, 2002; Cossette and Walker, 2007]. If developers provided the syntax describing relevant dependencies in their software system, it should be possible to generate useful change impact analysis tools to support them in proactively planning future changes. The limitation until now has been the costs of specifying this syntax.

In this chapter, I present a semi-automated, *lightweight* approximation of semantically-aware tool support that:

1. provides more effective dependency detection than existing manual, lexical-analysis techniques;

2. is adaptable to arbitrary language and technology combinations; and,

3. requires significantly less effort to construct than fully-automated approaches, while remaining easy to configure (and hence, is lightweight [Murphy and Notkin, 1996]).

My approach is embodied in the DSKETCH tool, which leverages a developer's knowledge about the syntax and semantics of the languages and technologies used in his software system to generate customized dependency analysis support. To use DSKETCH, a developer writes simple pattern specifications capturing relevant identifiers in each language, to which DSKETCH applies heuristics capable of cross-referencing these identifiers, thus identifying probable dependencies in the system. The result is displayed to the developer by annotating the source code[1] to indicate which pattern matched which identifiers.

DSKETCH is a plugin for the into the Eclipse integrated development environment (IDE), allowing a developer to continue to use their other tool support as desired. Developers can iterate on the pattern specifications to refine the set of dependencies until satisfied with the results. The workflow intended for the tool involves one developer configuring a set of patterns for a particular system, and allowing other developers to use these patterns in their tasks on the same (or similar) systems.

The rest of this chapter is organized as follows: in Section 4.1, I present the Sketch notation provided for developers to describe the dependency patterns in their software. I then explain how DSKETCH uses these patterns to detect identifiers in the source code in Section 4.2. Section 4.3 describes the heuristic approach for interrelating identifiers to suggest the presence of dependencies. The graphical interface for the tool, and a description of its usage, is presented in Section 4.4.

Details on how the DSKETCH tool was evaluated, and an analysis of its results, are presented in Section 4.5 and Section 4.6. A discussion of their implications follows in Section 4.3.[2]

[1] I use the term "source code" liberally, including XML files, configuration text files, etc. in its application

[2] Some of the contents of this chapter have been previously published [Cossette and Walker, 2010], and have been revised and expanded for integration into this dissertation. ©2010 ACM. This is a minor revision of the work published in the *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2010, http://doi.acm.org/10.1145/1882291.1882335. Included here by permission.

4.1 The Sketch Notation

My approach is based on the observation that the portion of a language's syntax or semantics which are necessary for identifying static, structural dependency relationships in source code is a small subset of the entire language's specification [Wong et al., 1995], and if developers could specify just those details for the languages in their software, this may be sufficient information to perform and analysis of where dependencies may exist. Previously, I tried applying island grammars to polylingual dependency analysis, but found that the costs of specifying dependency syntax was surprisingly high [Cossette and Walker, 2007]: the complexities of writing richer island grammars rose quickly, and began to require a depth of skill with grammar-based tooling (or, grammarware [Klint et al., 2005]) that an average developer may not have.[3]

The Sketch notation is a reflection of the lessons learned from my previous work in creating the LUTHER testbed for island grammar evaluation [Cossette and Walker, 2007], and provides a simplified means of specifying dependency syntax for a programming language that is usable by a developer after minimal training. The idea is that developers "sketch" patterns roughly conforming to the syntax needed to recognize relevant dependency syntax in their source code, in a manner both familiar to them, and of significantly less effort than a complete syntactic specification. The Sketch notation is also informed by concepts and ideas from previous research on fuzzy parsing [Koppler, 1997], and island grammars [Moonen, 2001, 2002].

There are four key design principles which shaped the development of the Sketch notation.

1. **Human readability is paramount.** A developer needs to be able to understand the specification easily if they are to clearly see cases when their pattern does not work, or if they are to reuse these patterns in other systems.

2. **Details that hinder the expression of intent should be eliminated.** Because the

[3]To be clear: many, if not most, developers could be capable of providing these specifications. But if they are not actively using grammarware on a day-to-day basis, using such tools will require extensive learning or relearning, and a significant up-front investment before any usable analysis can be written. I suggest this is an unrealistic demand on developer time in many cases.

developer is trying to describe key dependency syntax, the focus must be on letting them specify what is necessary to recognize, versus specifying what could possibly occur.

3. **Allow implicit and explicit ignoring of irrelevant code.** A key advantage of applying island grammars to dependency analysis is the ability to identify what parts of a language's syntax are irrelevant and can be ignored. But this still requires the developer to always specify what should be ignored. Instead, there should be an implicit understanding of cases in which the lack of specification implies that the syntax is not important.

4. **Use "anchor points" to help deal with ambiguity.** Often, certain keywords or symbols are expected to occur in contexts that are important to recognize (e.g., a (after the declaration of a method's name, but before the start of its parameter list, in JAVA). Explicitly looking for these points can help anchor patterns to specific cases, make it less likely the pattern unintentionally makes a mistaken match.

The sketch language comprises four main components.

1. **Identifier Keywords.** DSKETCH is primarily concerned with extracting identifiers from source code on which dependencies may be based. Two keywords are provided for this purpose: global and local. Each of these keywords matches a sequence of characters that lexically conforms to a C-style identifier. The different keywords are provided to allow the developer to give a very crude approximation for scope resolution, by indicating if dependencies involving this identifier should be sought across the developer's entire source code base (global), or only within the file (local) [Chen et al., 1990].

2. **Anchors.** Patterns can require specific lexical sequences to be embedded within them to improve their ability to discriminate, called *anchors*. Any string or char-

acter that serves as an anchor is enclosed in single quotes (e.g., 'public', or '{'). A backslash is used as an escape-character (i.e., '\\' matches a single backslash character and '\" matches a single quotation mark). Anchors allow developers to provide additional pattern information to improve match precisions, without confusing the tool as to what portions of the match need to be analyzed for dependency analysis.

3. **String Literals.** It is quite common for source code and important identifiers from one programming language to be embedded in the source code for another language, typically as a string literal [Halfond and Orso, 2005; Cossette and Walker, 2007]. In addition to anchor patterns described earlier, the developer can add the STRING_ prefix to the identifier keywords (e.g., STRING_global) to instruct DSKETCH to grab the entire contents of a string literal and treat it as having the scope of the suffixed identifier.

4. **Junk productions.** Some patterns can be difficult to specify in detail, because of the e various syntax combinations that may exist in the source code; this is particularly onerous when these details are extraneous to the purpose of the developer. Rather than anticipate all such combinations, the developer can use a *junk production* to instruct DSKETCH to ignore everything it finds at that point in the pattern [Moonen, 2001, 2002] The syntax of a junk production is [junk]; it must be followed by an anchor to give DSKETCH a concrete means of determining when the irrelevant material ends. The junk production accepts every character it encounters until it reaches the first match for the anchor.

4.1.1 Sketch Notation Example

As an example usage of the sketch language, Listing 4.1 presents an excerpt the source code of the OPENBRAVO ERP[4]. Listing 4.1 is from the MenuData.xsql file in the org.openbravo.erpCommon.

[4] http://www.openbravo.com/

utility package in OPENBRAVO, and is an XML file containing SQL queries that are intended to be mapped to JAVA objects.

```
1 <SqlClass name="MenuData" package="org.openbravo.erpCommon.utility">
```

Listing 4.1: Excerpt from MenuData.xsql

To match the identifier in the **package** attribute, the developer can sketch this pattern:

```
1 '<' 'SqlClass' [junk] 'package' '=' STRING_global
```

Listing 4.2: Sketch for MenuData.xsql

The anchors '<', 'SqlClass', 'package', and '=' are all used to describe the characteristics of an XML tag element, but also describe a specific tag element that has special meaning (in OPENBRAVO, the SqlClass tag defines the JAVA class to which the encapsulated SQL queries will be mapped at runtime). Here the developer is not interested in capturing the name (or any other) attribute, so he uses the [junk] production to skip over anything intervening, and then uses the STRING_global keyword to capture the contents of the string literal and mark it as having global scope.

4.2 Detecting Identifiers

DSKETCH compiles the developer's sketched patterns into a set of equivalent regular expressions. The tool traverses the source code, determines which patterns need to be applied to each source file based on its extension, and attempts to match the patterns. Matches for the identifier keywords and details of their position are stored within a plugin-hosted database. Once this process is completed for the entire code base, the database is analyzed to build a symbol table listing the unique identifiers found during the analysis, the relationships between the identifiers in the system, the contexts each identifier is found in, and which patterns discovered them.

4.2.1 Compiling Sketched Patterns

DSKETCH uses a parser I created using JavaCC[5] to analyze the patterns sketched by the developer, and to transform them into a set of regular expressions. The transformation rules are as follows:

- Occurrences of global and local are expanded into a regular expression appropriate for matching C-style identifiers. This regular expression is enclosed in parentheses to allow back-referencing to be used to recover the identifiers matched in the pattern.
- Occurrences of the STRING_-variant keywords are used to match the entire contents of a string literal, and then record its matches as a global or local identifier.
- Anchors are embedded into the regular expression, with appropriate escape-characters inserted when necessary to avoid their interpretation as meta-characters.
- A junk production (which must be succeeded by an anchor) is translated into a regular expression that permits arbitrary characters but that will greedily match the anchor.
- The presence of whitespace is dynamically calculated based on the combination of productions in the DSKETCH patterns. The rules are as follows:
  - No whitespace characters match at the start or end of the pattern.
  - One or more whitespace characters must match between any two consecutive identifier keywords.
  - No whitespace is matched between a [junk] production and its succeeding anchor.

[5] http://javacc.dev.java.net

- Between an anchor and any other element, zero or more whitespace characters may be present.[6]
- Between any two other elements, zero or more whitespace characters match.

DSKETCH then uses the regular expression engine[7] provided with the JAVA 6.0 Standard Edition software development kit (SDK) to apply these regular expressions to the developer's source code. For each file, all regular expressions appropriate to the programming language(s) contained therein are applied, and matches are extracted. For each match, DSKETCH stores:

- the match's location;
- the name of the pattern that found the match;
- the identifiers in the match tagged as global or local; and,
- in the case where multiple identifiers were located from the same match, annotates these identifiers as *siblings*.

4.2.2 Pattern Collisions

It is possible to sketch two or more patterns that match the same portions of a file; perhaps the sketch is wrong, or it may reflect the difficulty of writing patterns for that programming language's syntax. Such a *pattern collision* may cause the same identifier in the source code to be recognized by multiple patterns. I cope with pattern collisions in two ways.

1. In cases where one pattern marks an identifier as local, and another as global, I prefer the pattern which recognized the identifier as global to reduce the risk of false negatives in the analysis.

[6] This can cause a false match in some cases (e.g., 'public''void' matches publicvoid), but I feel that this situation would occur rarely in practice.

[7] http://java.sun.com/javase/6/docs/api/java/util/regex/package-summary.html

2. If multiple patterns recognize the same identifier with the same scope (e.g., all global), I associate the identifier with the first pattern match.[8]

4.3 Detecting Dependencies

After extracting a set of identifiers using the sketch patterns, DSKETCH uses heuristics to approximate how these identifiers indicate dependencies in the system. The assumptions that DSKETCH makes about how dependencies are formed reflects both my prior attempts to provide a simplified notation scheme for the developer to use, and my own experiences in dealing with polylingual software systems. The following heuristics are currently used by DSKETCH:

- Dependencies can only exist between identifiers extracted from source code through one of the developer's patterns. That is, DSKETCH does not search the source code to see if matches can be found outside of the pattern set the developer provides.
- Dependencies must be lexically identical, ignoring case.
- If the identifier was classified as global, it is dependent on all other occurrences of that identifier, regardless if they were marked as global or local.
- If the identifier is classified as local, it is dependent on all other occurrences of that identifier which:
    1. are marked as being local; and,
    2. are discovered within the same source file.

These heuristics derive from the observation that many polylingual technologies rely on lexical identity as the primary means of cross-referencing declarations to references.

[8] In earlier iterations, I merged the meta-data associated with each pattern collision, and showed the developer all the patterns which matched a particular identifier. This seemed to confuse them more than it helped.

4.4 Using DSKETCH

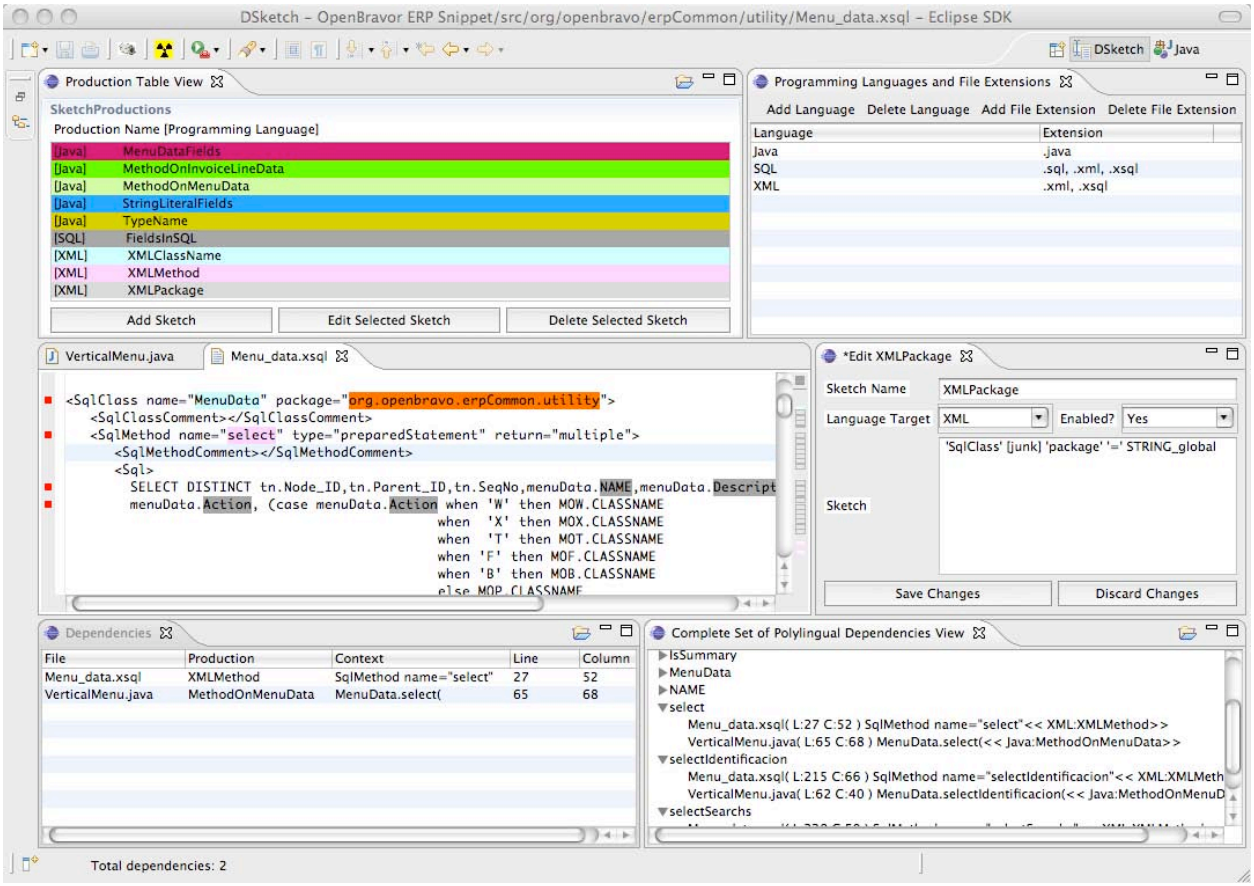

Figure 4.1: Screenshot of DSKETCH.

Figure 4.1 shows a screenshot of the DSKETCH dependency analysis tool, installed as a plugin to the ECLIPSE IDE. DSKETCH is intended for use by developers looking to investigate dependency relationships, particularly in a polylingual system, where a reasonable approximation of the dependencies is sufficient for their needs. A developer using the plugin will have a project in ECLIPSE with the complete set of source code for the software she intends to analyze.

The developer begins with a view (perhaps empty) of the sketches currently defined for her system, as shown by the Production Table view, as highlighted in Figure 4.2. For each Sketch, the Production Table view lists the developer-provided name of the pattern (to help her remember what dependency syntax it matches), the programming language it should be applied to, and the colour used to identifying dependencies matched with this Sketch. From here, she can add a new

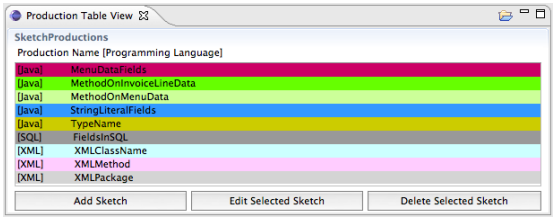

Figure 4.2: Sketch list.

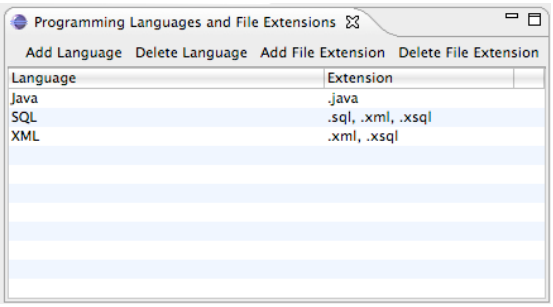

Figure 4.3: Language mappings.

Sketch to the set of productions, edit the Sketch for a pattern to refine it, or remove a Sketch from her collection of productions. By double clicking on an individual Sketch, she can enable or disable its involvement in identifying dependency relationships; this capability has proved itself valuable in debugging sketched patterns. Each Sketch is associated with a specific programming language; the developer defines which file extensions can contain which programming languages in the language map view (shown in Figure 4.3), so that DSketch knows where different Sketches need to be applied.

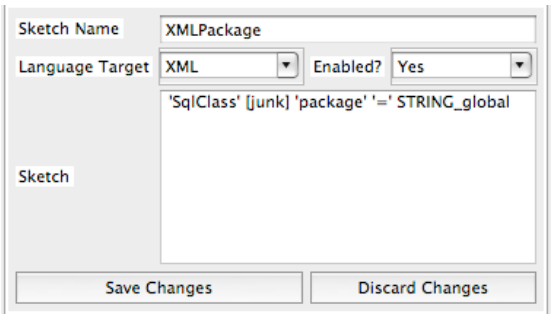

Figure 4.4: Creating or Editing a Sketch pattern.

To create a Sketch, the developer clicks on the "Add Sketch" button; editing a sketch requires she first select the desired Sketch, and press the "Edit Selected Sketch" button instead. This will open the Sketch editor window, shown in Figure 4.4, which allows her to write her Sketch pattern, assign it a meaningful name to describe the dependency syntax it recognizes, and choose the programming language it is intended to be applied to. She also has the option of choosing to have the pattern enabled or disabled, depending on whether she wants to see its analysis applied immediately or not. When she is satisfied with her creation (or changes), she presses the "Save Changes"

button. If she decides to abandon the creation or edit, she can discard all of the information with the "Discard Changes" button.

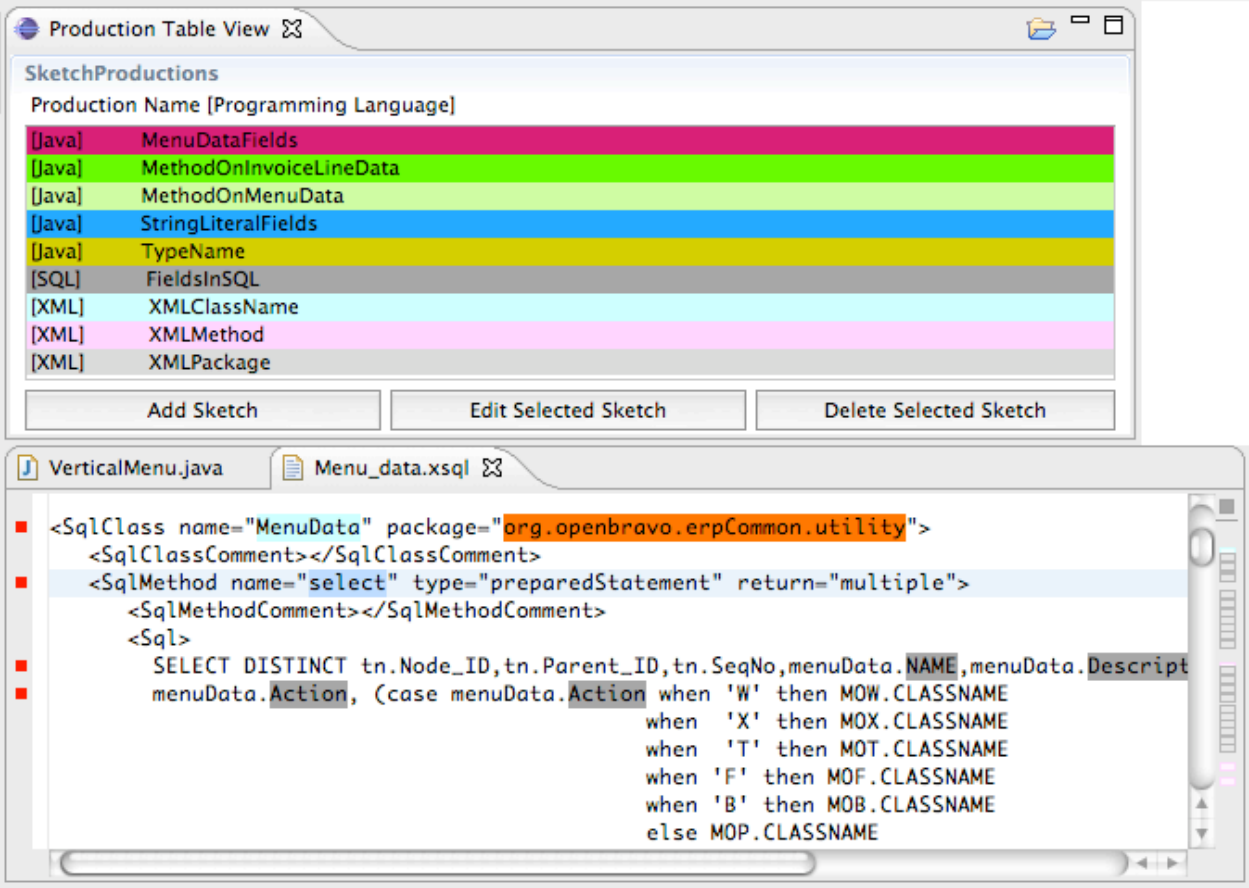

Figure 4.5: Dependency highlighting in source code.

To see where points of dependency may exist in a file, the developer can use a right-click context menu in her source code editor to highlight all of the pattern matches in the current source code file, as shown in Figure 4.5. DSKETCH then highlights identifiers in the file, colour-coding them to indicate which Sketch pattern matches at that location, as well as setting a tooltip on the identifier with the same information. For example, in Figure 4.5, the "XMLMethod" sketch has extracted an identifier from the name attribute in the SqlMethod tag, and has marked the identifier with the colour pink to indicate (1) that it has been matched, and (2) that the "XMLMethod" sketch has identified it as relevant.

If the developer right-clicks on the "select" identifier, she can use a context menu to have

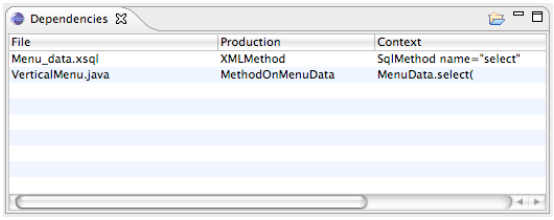

Figure 4.6: Detected dependencies.

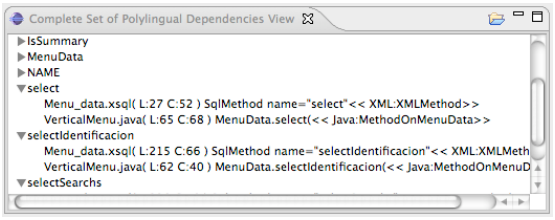

Figure 4.7: All dependencies.

DSKETCH display polylingual dependencies on this identifier (see Figure 4.6), which here shows a match to a static method invocation in JAVA. She can also view the entire set of dependency predictions for her system if she wishes to have a sense of the entire scope (see Figure 4.7). In either view, each entry shows the identifier matched, its location, and the context surrounding the match. She can leverage this information to initially discriminate between the dependencies, and to find those dependencies that are more relevant for her investigation task. If the developer sees a dependency that she wishes to explore, she can double-click on it to automatically navigate to the appropriate location in a source code editor.

4.5 Evaluation

The intent of DSKETCH is to provide developers with adaptable polylingual dependency analysis tool support that is relatively easy to configure for a system, and more effective than state-of-the-practice approaches. I conducted a qualitative investigation of the tool to determine whether it meets these goals. My specific research questions were as follows.

RQ 4.1

Can developers successfully configure the tool for dependency analysis?

RQ 4.2

How much effort is required to adapt the tool to a new system?

RQ 4.3

Is the tool effective for polylingual dependency analysis?

RQ 4.4

How does the approach conform with or contradict developers' expectations?

To address these questions, I conducted two case studies in which industrial developers used DSKETCH to perform dependency analysis in two separate software systems.

4.5.1 Case Study Overview

Two industrial software developers were recruited to participate in the study. Each of the participants was asked to investigate the polylingual dependency relationships between two source code files in a particular system.

Case Study 1 (see Section 4.5.2) focused on the open-source OPENBRAVO ERP system[9], an open-source, web-based, enterprise management system that comprises more than 3,000 files written in JAVA, HTML, XML, and SQL, with a combined size of over 300,000 lines of code. Case Study 2 (see Section 4.5.3) involved the IBATIS JPETSTORE V5.0 reference implementation.[10]

Both of these software systems use a data-mapping framework to allow JAVA source code to interact with SQL queries and results, by embedding SQL queries within XML tags that describe how JAVA code should interact with them. However, each uses a different technology with similar yet distinct semantics for facilitating these operations: OPENBRAVO uses (in part) a compiler called SQLC[11], while JPETSTORE uses the IBATIS DATA MAPPER framework.[12]

Each case study was structured as follows: After a short tutorial explaining the mechanisms that the particular system uses to map dependencies across these languages, the participants were asked to manually investigate a portion of the source code of the system and determine what were the polylingual dependencies present. Each was provided with the ECLIPSE IDE, v3.5. The goal was to have the participant become sufficiently familiar with the system that they would be able to later judge the efficacy of their DSKETCH configuration in detecting these same dependencies.

[9] www.openbravo.com
[10] ibatis.apache.org/java.cgi
[11] http://wiki.openbravo.com/wiki/ERP/2.50/Developers_Guide/Concepts/SQLC
[12] ibatis.apache.org

Since the developers were unfamiliar with the details of the particular systems under analysis, they were provided with training material and permitted to ask me questions concerning dependency mechanics over the course of the study, as a means of simulating the experience each would have when working with their own systems.

After the developer declared himself as being done with the manual investigation task, he was presented with the tool. After a training session in which the tool's operation, Sketch language, and behaviour were explained, each participant was given time to experiment with the tool on a piece of sample code to become familiar with writing patterns. Once he felt comfortable with the tool, he was given his assigned system to investigate. After their investigation was complete, I interviewed the participant about his experiences with DSKETCH.

4.5.2 Case Study 1

The participant here described himself as having 16 years of industrial experience as a software developer, familiar with maintenance of polylingual software systems, and for the last several years as working as a principal developer and architect on a polylingual software development kit used extensively in his company's and clients' products.

To constrain the task size, I asked the participant to manually investigate all the polylingual dependencies between just two files in OPENBRAVO: VerticalMenu.java and MenuData.xsql from the org.openbravo.erpCommon.utility package. These two files comprise 429 non-commented lines of code and extensively reference each other. After configuring the tool to his satisfaction, he was supplied with two additional files from the system InvoiceLine.java and InvoiceLine_data.xsql in the org.openbravo.erpCommon.info package), and asked to adjust his configuration to recognize dependencies between them.

4.5.2.1 Manual Treatment

After being given demonstrations about the dependencies on some example files, the participant commented that he was accustomed to using SQL/XML data mapping frameworks, but as each

framework is slightly different, he was primarily concerned with understanding how the semantics of this particular technology worked. The participant started his investigation by examining the relationship between the XML tags and the JAVA code; he found tracing the various SQL queries and enclosing XML tags painstaking in the default text editor, and opened the MenuData.xsql file in ECLIPSE's XML editor to browse the contents as a tree structure.[13] With this view in hand, he easily determined the mapping of the MENUDATA table and its associated queries to the appropriate type static method invocations in JAVA.

The participant continued with the laborious task of identifying where SQL column references in the MENUDATA table were being accessed as fields on MenuData-typed objects, remarking at one point: "Is there an easier way to find all of these field names?" He noticed that a particular coding convention caused MenuData declarations to be labelled as "menuData", and he proceeded to perform lexical searches on this identifier to find the related field accesses. In doing so, the participant missed some occurrences elsewhere that failed to follow this naming convention (he did not discover this issue at the time). He then declared himself finished, having spent roughly 40 minutes in the investigation.

4.5.2.2 DSKETCH Treatment

I then commenced the training of the participant in the use of DSKETCH, including practice at sketching patterns and interpreting the results. During training, two issues arose.

1. The participant asked if regular expression closure operators (e.g., +, $*$) are supported; they currently are not.

2. The notion of global versus local identifiers caused some confusion. When sketching a pattern to recognize the value associated with an XML attribute, he debated as to which scope made more sense: the XML language defines that the value of an attribute only has meaning within the context of its enclosing XML tag element, which would imply that the value should be marked as having a local meaning.

[13] The file extension had to be changed to work around the editor's constraints.

However, OPENBRAVO uses the value associated with the name attribute in certain tags to define which static method invocation the enclosed SQL query should be mapped to, indicating that this attribute value has a meaning outside of the file, and should be marked as global; the participant eventually recognized this.

The case study was suspended at this point at the participant's request, and was resumed the following day. The participant spent approximately 15 minutes reviewing the test patterns sketched the previous day. He then chose to delete these patterns before starting the configuration task.

The participant started by sketching three patterns quickly (one for each of the languages in use, to detect a particular cross-language dependency), and testing them against the VerticalMenu class. In general, this participant tended to pick a particular snippet of code that was typical of the dependency characteristic he wanted to recognize, and to sketch a pattern designed to specifically recognize that snippet. Upon reviewing the results, he would then refine that pattern if it was too specific, or not specific enough, to match other locations in the source code until he was satisfied with the pattern's coverage. Primarily, the participant wrote patterns to recognize table and column references in SQL, attribute values associated with XML tags which encapsulated SQL queries, static method invocations on the MenuData class in JAVA, and field accesses in JAVA.

At one point, the participant noticed that one of his XML sketches was not working: he was attempting to capture the **package** attribute value in the SqlClass tag, but because another attribute preceded it the lexical signature of the tag did not match his sketch. After realizing that he wanted to ignore the intervening attribute(s) in these cases, he inserted the [junk] production into his sketch.

The participant was then given the second set of files from the OPENBRAVO system, and asked to see what adjustments needed to be made to recognize dependencies in this files. The patterns written for the SQL and XML code did not need to change, but his set of JAVA patterns needed to change in two ways:

1. The participant originally sketched a pattern to recognize static method invocations specifically on the MethodData class, as these represented SQL queries; in the new

files, similar support had to be added for the InvoiceLineData type.

2. The InvoiceLine class has numerous string literals that contain references to SQL tables and columns, a feature not present in the previous file set.

After making these modifications, he was satisfied with the results. He had spent approximately 30 minutes configuring DSKETCH for the first set of files, and an additional 10 minutes to adapt the patterns to the second set of files.

4.5.2.3 Observations on Workflow, Success, and Effort

The participant tended to iteratively supplement his configuration by sketching a new pattern, testing it, and adjusting it as needed before moving to the next. He worked through one language at a time, capturing all the relevant syntax before moving on. He also tended to sketch his patterns as narrowly as possible; he said that his attitude was "Garbage-in, garbage out. A tool is only as good as its inputs."

Over time, the participant seemed to gain greater trust with the tool as he investigated predicted matches. In the first set of files, he noticed that DSKETCH did not find a match for one of the SQL queries. After investigation, he realized that this omission was correct—that query was never accessed in the JAVA code. Later, a similar issue occurred with the second set of files, but he now accepted the tool's predictions without feeling the need to investigate.

All the sketch language features were used at some point by the participant, and with one exception were found useful. During the study, the STRING_-variant keywords were found to have an unintended consequence: these keywords match the contents of a string literal but implicitly assumes that the string is *not* empty. The presence of several empty strings in the second set of files caused the resulting pattern to match content between the last double quotation mark (") in an empty string, and the first occurrence of a double quotation mark later in the file. The participant easily fixed this by eliminating the STRING_-variant of the keyword with its normal variant and appropriate anchor productions.

4.5.3 Case Study 2

The second participant described himself as having 7 years of industrial experience as a software developer, the vast majority of that involving polylingual systems. I asked the participant to manually investigate the change impact of altering the user authentication mechanism in JPETSTORE. To limit the required effort to a reasonable level for a case study, I restricted his investigation to just the JAVA, XML, and SQL portions of the system.

After the manual investigation, he was trained on DSKETCH, and provided with the configuration generated by the Case Study 1 participant as a starting-point configuration for JPETSTORE.

4.5.3.1 Manual Treatment

During the explanation of how IBATIS maps data objects in JAVA, the participant commented that he was familiar with this paradigm as the semantics seemed very similar to those used by the HIBERNATE data mapping framework,[14] which he had used before.

The participant constructed a change plan on paper to record his understanding of how the system would be affected by the anticipated change. He largely investigated the system by opening source code files, and scanning for relevant portions of code, pausing in some cases to thoroughly understand a particular segment that seemed to be of special significance to the authentication mechanisms. After some initial exploratory searching in this fashion, he returned to the Account.xml file which contained most of the SQL queries involved in the user authentication mechanism in JPETSTORE, and looked for dependencies emanating from the SIGNON table throughout the code.

The participant would either use lexical searches, or ECLIPSE's JAVA tooling, to follow dependency links in the code, but occasionally would investigate a new file when its name or package identifier suggested it was of relevance. He continued this pattern of investigation until he reached the AccountBean class which interacts with the system's presentation layer, after which he finalized his change plan and declared himself finished. He had spent roughly an hour creating this change plan.

[14] http://www.hibernate.org/

4.5.3.2 DSKETCH Treatment

I then commenced training the participant on using DSKETCH, with the final configuration from Case Study 1 provided as a starting point. During training, he asked if the zero-or-one regular expression operator (i.e., "?") was supported; it currently is not. Once the participant felt comfortable with the tool, he began his configuration task for JPETSTORE, again with the patterns from Case Study 1 as a starting point.

The participant began by systematically examining the patterns, and had no problems reasoning about the intent behind the JAVA and SQL patterns. This changed when he came across the first of the XML patterns: after examining the pattern, he was confused about the intent as it described syntax that he did not recognize in their system. He manually examined the JPETSTORE code to look for relevant examples, and only after a few minutes of this did he remember that these patterns had been created to match dependency syntax which differed from that used by IBATIS in JPETSTORE. He decided that the XML patterns were not useful for his context, but the existing JAVA and SQL patterns could be reused (even though at least one pattern described syntax which did not occur in JPETSTORE); surprisingly, he decided to not delete the irrelevant patterns since they seem to have no ill effect.

He continued his configuration of DSKETCH by applying the patterns to see what dependency syntax was missed. In the case of the SQL code, he noticed that the existing patterns captured most of the syntax he was interested in, but missed a few small cases where queries did not use explicit access of a table column (e.g., SIGNON.USERNAME) that the other patterns had leveraged. He decided in this case that the tool's configuration was sufficient since it already found relevant matches on those tables and columns in the same file, and felt it was unnecessary to write specialized patterns to capture these few cases. He also decided that only one additional JAVA pattern was needed so that field declarations were recognized, so that DSKETCH could match them against the SQL column names to which they were mapped.

In exploring the resulting matches, he discovered two false positives which he had not expected:

DSKETCH indicated that there were dependency matches on the USERNAME and PASSWORD columns in the sql−map−config.xml file, which otherwise is expected to simply list which XML files contain SQL query mapping data for iBatis. DSKETCH's dependency preview indicated the context of these matches as being JDBC.Username and JDBC.Password, which was ambiguous: it could describe a JAVA field access, or a SQL table/column pair. After investigating the match location, the participant saw that these were actually describing property values which were to be supplied by the IBATIS system for connecting to the JPETSTORE database.

4.5.3.3 Observations on Workflow, Success, and Effort

The participant's workflow was substantially different than that of the Case Study 1 participant, because he intended to formulate a specific action plan for enacting a change to the system. Consequently, he spent a considerable amount of time attempting to understand how existing code functioned, rather than simply tracing dependency connections. Having an explicit change plan may also have explained why he did not feel it necessary to configure DSKETCH to recognize all the known polylingual dependencies in the system: those few dependencies were easier to simply make note of in the change plan, rather than invest additional effort to address. The participant felt comfortable reusing the patterns written by the earlier participant, and only needed to make minor alterations for his system's context.

In using DSKETCH's dependency preview window, the participant noted that the context provided for matches was insufficient: "Having the filename and the context helps, but I need to see the entire context." The context currently provided by the tool is the entire match (including anchor and junk productions) for that pattern, which in some cases he encountered was too limited to provide useful feedback. Had the preview included most of the code in that line, it would have been sufficient information to reject the dependency without needing to investigate the code snippet.

4.6 Analysis of Observations

I will now revisit the research questions posed in Section 4.5, in light of these observations.

4.6.1 RQ 4.1: Can developers configure the tool?

Both participants were able to configure DSKETCH tool to recognize polylingual dependencies in systems, despite a limited amount of exposure to the tool prior to use, and in spite of not being experts with the systems they were analyzing. In the case of the first study participant, he was able to return to his task a day later, revisit his previous work, and continue his configuration of DSKETCH within a 15-minute period. The second participant was able to take a set of patterns written by a different developer for a different system, understand the intent behind the patterns, and supplement them appropriately for his own work.

4.6.2 RQ 4.2: How much effort is required to adapt?

Both participants described the effort as non-trivial, but reasonable. The second participant was able to easily adapt the DSKETCH configuration for OPENBRAVO to the JPETSTORE system, by reasoning about what features the existing patterns were intended to match. He discovered which patterns were inappropriate, and appropriately supplemented the configuration with additional patterns to address what was lacking. The developer did note though that it was difficult to get an initial sense of what was missing in the pattern configurations; if the pattern set were sufficiently large, it could be difficult to understand the tool's effective coverage. The first participant commented that: "[DSKETCH] has an up-front cost that has to be paid in learning the patterns and writing them, but it's reasonable. And once written, they're easily applied elsewhere."

Both participants indicated that the Sketch notation was far easier to work with than regular expressions; the first participant pointedly commented that "Regular expressions [are not sufficiently human-usable]."[15]

4.6.3 RQ 4.3: Does the tool produce results that are reasonably correct?

Both participants indicated that the tool was reasonably correct: the first study participant thought he had detected a false negative, but on further investigation it turned out the tool was correct. The

[15] I have suppressed his more "colourful" phrasing.

second participant noted there were a few false positives, but he did not notice any false negatives outside those cases he decided were not worth capturing.

Both participants felt the tool was more effective for polylingual dependency analysis than the manual approaches they tried, especially because they were not aware of any other alternatives. But they would not blindly trust its predictions, needing some manual investigation of the results. In its current state, it dramatically reduces the number of false positive and false negative results they normally deal with in such investigations, saving them considerable time. The participants both suggested some additional constructs to improve the expressiveness of the pattern language, including the ability to combine patterns using set operations (e.g., intersection).

4.6.4 RQ 4.4: Does the approach work?

How the developers configured the tool varied: the first participant tended to write highly tailored and precise patterns which attempted to match dependency features as specifically as possible, while the second participant was more flexible in his matches and more willing to ignore cases which were deemed to be too much work to catch, or to ignore refining existing patterns if they were close enough. Their attitudes towards configuring the tool reflected their tasks, and the tool seemed amenable to both approaches.

4.7 Discussion

4.7.1 Qualitative vs. Quantitative Evaluation

The case study evaluating DSKETCH is qualitative in nature, focusing on reporting the participants' perceptions of the effort required to configure the tool, and of the tool's prediction effectiveness for their case study task. While some quantitative values are reported (for example, the time participants spent in training, or in configuring DSketch), their values are meant to describe their actions, rather than report on their performance.

The research questions in this study are exploratory in nature, and concern whether or not the

approach is useful and viable. Participants in this study were exposed to a new pattern notation, new software systems and technologies, and a new tool for the purposes of dependency analysis. By emphasizing the qualitative nature of the case studies, focus can be drawn to identifying the strengths and weaknesses of the approach; only after these results are taken into account in the tool design will a formal and quantitative experiment be appropriate [Murphy et al., 1999]. Future research applying these findings through improvements in notation, heuristics, or other factors over DSKETCH will thus need quantitative studies to show the benefit of their contribution.

4.7.2 Improving pattern expressiveness

The study participants identified a few additional notations or capabilities to allow writing more expressive or specific patterns. Some of these limitations, specifically the lack of regular expression closure support, were due to limitations I encountered when trying to use back-referencing in the patterns to extract matches in these cases (only the last back-referenced match in such patterns are captured). Resolving these issues may require adopting a different technology for applying the patterns, or creating specialized support for these cases.

4.7.3 Usability and performance

Both participants suggested modifications to the interface to adapt it more succinctly to their desired workflow: the first participant wished for more extensive pattern debugging capabilities, the second expressed a strong desire to see the tool even more tightly integrated into the ECLIPSE IDE such that it would resemble the existing JAVA tooling support in operation and usage; both indicated that it would be desirable to support their tendency to choose a particular code snippet as exemplar of the pattern they are trying to write.

Performance is also an issue: on the JPETSTORE system, the second participant noted that the tool paused notably during its analysis. Since his preferred workflow involved iteratively refining and applying patterns to the system, it was important to him that the speed of the tool be as close to real-time as possible.

4.7.4 Adapting to large-scale system issues

As the systems under analysis by DSKETCH grow in size, I expect (and my participants expressed similar concerns) that it will become more difficult to keep track of which patterns are contributing effectively to the tool configuration. The likelihood of the tool's confusing dependency matches due to its simplified notation language and heuristics will increase, leading to more false positives in the developer results.

Rather than put additional onus on the developer to supply increasingly detailed configuration information, I would like to explore mechanisms that could allow the tool to adapt itself and its heuristics to the systems it analyzes. One possibility is leveraging feedback from the developer as to the effectiveness of its predictions as a means of learning, or providing the developer feedback as to which of their patterns (and why) are proving effective or ineffective in their investigation of the system.

4.8 Summary: Proactive Change

DSKETCH provides a lightweight approximation of semantically-aware tool support for polylingual dependency analysis. Rather than attempting to encode knowledge of languages and cross-language technologies in a general, automated fashion, DSKETCH provides semi-automated support that relies on developers to configure the tool for the languages and technologies in their software. A lightweight Sketch notation is provided to describe the relevant dependency syntax in each language, and from these Sketch patterns DSKETCH generates polylingual dependency analysis tool support specialized for the developer's system. Developers can iteratively refine their dependency Sketches until they are satisfied with the results, and can share these patterns with other developers working on the same, or similar software systems.

Case studies with industrial participants showed developers could configure DSKETCH reasonably well with a brief period of training for polylingual dependency analysis, and were able to adapt the configuration of the tool to new situations easily. Further, it was possible for developers

to understand and appropriately reuse configurations provided by others, revising and extending until suitable for the target system. Participants successfully used the tool to detect dependencies in their systems, and were satisfied with the effectiveness of its predictions: developers stopped refining their dependency configuration further when they tool was effective enough that a minor amount of additional manual exploration was sufficient to complete the task.

Chapter 5

The Evolution and Migration of Software Libraries

Software developers rely on external libraries to provide reusable functionality. Ideally, developers can rely solely on the application programming interface (API) that libraries provide to access their functionality, without worrying about their implementation details [Morel and Alexander, 2004]; they hope that the API is a contract that will not change [Chow and Notkin, 1996; Larman, 2001; Henkel and Diwan, 2005; Bloch, 2006]. However, software libraries are susceptible to the same environmental pressures to change that all software systems face [Lehman, 1980; Parnas, 1994; Perkins, 2005]. As a result, library maintainers need to evolve their systems in ways that sometimes result in an incompatibility between the old and new versions of their API [Henkel and Diwan, 2005; Perkins, 2005]. This raises a dilemma for developers whose software is a client of the library: when they migrate to the new API version, the changes the library developers made will now impact their software, and must be redressed.

Much research has addressed the issues involved in both identifying adaptations to accommodate API changes, and enacting library migrations [Chow and Notkin, 1996; Godfrey and Zou, 2005; Henkel and Diwan, 2005; Dig et al., 2006; Xing and Stroulia, 2007b; Kim et al., 2007; Dig et al., 2008; Nita and Notkin, 2010; Wu et al., 2010; Kapur et al., 2010; Dagenais and Robillard, 2011]. Yet only a few papers empirically study how APIs change over time, classifying the nature of the changes these systems undergo [Dig and Johnson, 2006b; Kapur et al., 2010]. Despite numerous evaluations of change recommendation or library migration approaches, there exists no established corpus of unbiased library change data. To both fully appreciate the complexities of real library migration issues, and to serve as a standardized basis for evaluating novel solutions, a collection of all points of breaking change between a set of API versions is need. Such a corpus of library change data should also identify the correct replacement to migrate dependent code in

each case. This would permit absolute measurements of the quality of recommendations, including knowledge of misses—important goals for this growing research area.

In this chapter, I present an empirical investigation into the changes of Java software libraries and their APIs across multiple revisions. For each pair of library versions involved in a single migration, I identify all points of change in the new version of the API that resulted in a *binary incompatibility* with respect to its previous version. A binary incompatibility (BI) is defined as any change to a type such that code compiled and linked without error to a binary containing that type prior to the change, does not link to a new binary containing the changed type [Gosling et al., 2012, ch. 13]. For each BI:

1. I determined whether replacement functionality exists in the new library version;

2. I manually applied six change inference techniques to each BI, to observe the ability of each to recommend correct replacements; and,

3. I analyzed each of the replacements found, to classify the nature of the transformation that would need to be applied to client source code to adapt to that particular library change.

The rest of this chapter is organized as follows: in Section 5.1, I lay out the methodology used to investigate each of these libraries for API changes, how each change was classified, and how I manually applied existing inference techniques to evaluate their performance. I then present the results of the empirical study in Section 5.2. A discussion of the implications of the results follows in Section 5.5.[1]

[1]Some of the contents of this chapter have been previously published [Cossette and Walker, 2012], and have been revised and expanded for integration into this dissertation. ©2012 ACM. This is a minor revision of the work published in the *Proceedings of the Twentieth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2012, http://doi.acm.org/10.1145/2393596.2393661. Included here by permission.

5.1 Methodology

My goal in this work was to create a corpus of API change data which describes what has changed in an API, and what it has been replaced with in the new version, with sufficient completeness and correctness to be useful to researchers to evaluate the effectiveness and limitations of API migration approaches. I was also interested in understanding how changes in libraries potentially impacted the effectiveness of techniques that are intended to detect such changes, or to adapt source code to them.

I chose to analyze five JAVA-based APIs:

APACHE STRUTS[2] versions 1.0.2, 1.1.0, 1.2.4, 1.2.6, 1.2.7, 1.2.8, 1.2.9;

LOG4J[3] versions 1.0.4, 1.1.3, 1.2.1, 1.2.2, 1.2.5, 1.2.6, 1.2.7, 1.2.8, 1.2.9;

JDOM[4] versions 1.0.b6, 1.0.b7, 1.0.b8, 1.0.b9;

DBCP[5] versions 1.0.0, 1.1.0, 1.2.0, 1.2.1, 1.2.2, 1.3.0, 1.4.0; and,

SLF4J[6] versions 1.0.1, 1.5.1, 1.5.5, 1.5.8.

I chose the STRUTS, LOG4J, and JDOM libraries to be subjects for this study, as it would allow me to attempt to replicate previous library migration studies. Specifically:

- Dig and Johnson [2006b] included the APACHE STRUTS in their empirical study, and both Schäfer et al. [2008] and Wu et al. [2010] used it in their evaluations;
- Dig and Johnson [2006b] and Kapur et al. [2010] both used the LOG4J library in their empirical studies; and,
- Kapur et al. [2010] used the JDOM library in their empirical study.

[2] http://struts.apache.org/
[3] http://logging.apache.org/log4j/1.2/
[4] http://www.jdom.org/
[5] http://commons.apache.org/proper/commons-dbcp/
[6] http://www.slf4j.org/

The choice of the DBCP and SLF4J libraries was influenced by their popularity [Kapur, 2010].

In addition to basic data collection about how these libraries changed, I sought to answer two research questions for each change.

RQ 5.1 How effective are recommendation techniques in finding the appropriate replacement for each change?

RQ 5.2 How feasible is it to mechanically transform client source code to automatically migrate each existing dependency to its replacement?

5.1.1 Determining Points of Change

To determine the changes within each library, I examined the binary incompatibilities introduced between successive versions of the same library. As mentioned at the start of this chapter, a binary incompatibility (BI) is defined [Gosling et al., 2012, ch. 13] as any change to a type such that code compiled and linked without error to a binary containing that type prior to the change, does not link to a new binary containing the changed type. The notion of binary incompatibility helps us to distinguish between all library changes, and those which directly impact software reliant on the library's API. For example, removing JAVA methods and fields marked as **private** from a library would not alter its API, as such entities are access restricted and cannot be used outside their declaring class. But renaming a **public** method, or changing the type of a field that was part of the API, would alter the API in a non-backwards compatible, and binary incompatible way. Lists of the different kinds of BIs which can occur, as well as some examples, are widely available online.[7]

The use of BIs offers several advantages over what has previously been attempted: the determination of what is and is not a BI can be mechanically determined from an analysis of the JAR files of two library versions, with a negligible risk of missing a breaking change between the versions; it does not rely on the correctness and completeness of documentation [Dig and Johnson, 2006b; Weißgerber and Diehl, 2006]; and finally, the introduction of a BI into a new library version

[7] e.g., http://wiki.eclipse.org/Evolving_Java-based_APIs_2

represents a point of change that *must* be addressed in client code to migrate to the new library. Binary incompatibility thus represents, in the general case, a necessary but not sufficient condition to overcome in migrating between libraries.

To determine the BIs between two library versions, I built an analysis tool based on CLIRR, an open-source tool for analyzing precompiled API JAR files for binary compatibility.[8] For each adjacent pair of library versions, I used the tool to determine the entities (classes, methods, and fields) in the old API which were binary incompatible with the new API, thereby generating a report listing CLIRR's classification of each. The final report for each API version transition was written to a file in comma-separated-value format, and imported into MICROSOFT EXCEL.

5.1.2 Finding Replacement Functionality

For each entity I found in the old API manifesting a BI with the new API, I examined the new API to determine if it contained an appropriate replacement. To assist my investigation, I systematically applied analysis techniques that have been proposed in the various change recommendation techniques described in Section 3.3.2 and Section 3.3.3. Each technique was applied independently of the others to evaluate its ability, in isolation, to find the correct solution for each BI.

Once all techniques were applied, if I had found a suitable replacement for the BI entity, I recorded it as the replacement in the corpus. In cases where I found an answer, but had doubts as to whether it was correct, I marked the replacement as "unsure". In a few situations, I found multiple potential replacements.[9] The dilemma in these cases was whether to record the recommended replacement, or to choose the replacement which was the easiest to migrate client code for. I decided that, in all such cases, I would record as the replacement the entity for which adaption of the client was easiest. Finally, for those cases in which I failed to find a replacement, I recorded the entity as "deleted".

[8] http://clirr.sourceforge.net; v0.7 was used.

[9] This occurred most often in cases where the old entity was marked as deprecated, it had an associated code comment that indicated what its replacement in the new API would be. While it later appeared to have been removed from the new API version, I would later discover that it had in fact been moved to another location, and was being used (while still deprecated!) by the new API.

I chose to apply seven change recommendation techniques manually, using only the tooling available in the ECLIPSE IDE.[10] I describe these below.

5.1.2.1 Original code comments

I examined the code comment associated with the binary-incompatible entity to see if it had a deprecated tag, and/or if the comment indicated what the replacement functionality should be. I then investigated the comment's recommendation to see if it was an appropriate replacement: this step was necessary, as I found a number of cases where deprecated tags referred to entities that did not exist in the new API. Dig and Johnson [2006b] also used as similar approach to this.

5.1.2.2 Release documents

I examined the release notes (if available), and any other documentation I could find online to see if the BI had been documented, and if its replacement was specified. Similar approaches are used in the literature [Dig and Johnson, 2006b; Weißgerber and Diehl, 2006] for collecting change sets.

5.1.2.3 Implementation details and dataflow analysis

I applied a series of investigative techniques on the code to determine if it was possible to reason about what the replacement for the code should be. Primarily, this entailed three things:

1. studying the implementation of the entity, as well as its usage, to reason about the intent behind the functionality;

2. performing static dependency analysis on the entity, and any other types, methods, or fields referenced by it, including constants [Verbaere et al., 2006]; and,

3. examining the inheritance hierarchy and subclasses of the containing class.

This approach took considerable time and effort to apply.

[10] Version 3.7 was used.

5.1.2.4 Call analysis

The call analysis technique examines references to the binary-incompatible entity in the old API version, and looks to see if those references also exist in the new API: if they exist, the entity or entities to which those references now refer to in the new API may be the replacement for the BI [Dagenais and Robillard, 2011]. For this technique, I followed the following sequence of steps.

1. I began by used ECLIPSE's static analysis tooling to find references to each binary-incompatible entity in the older version of the library.

2. I then checked if those references also existed in the new version of the library.

3. If they did exist in the new version, I examined the entity that these references pointed to in the new API to see if it would be an appropriate change recommendation.

4. If none of the references existed in the new library version, I then searched outwards via transitive references to the binary-incompatible entity, until either I had found a relevant recommendation or no more transitive references existed.

Similar approaches are reported in the literature [Schäfer et al., 2008; Dagenais and Robillard, 2011; Wu et al., 2010]; however, unlike these, I did not restrict the analysis to methods.

5.1.2.5 Lexical search via the code comment

I selected the first line of text in the entity's code comment (if any such existed), and performed a text-based search in ECLIPSE against all JAVA files in the new library. I examined all the code comment matches to determine if they were also associated with an entity that was the correct replacement.

5.1.2.6 Lexical search via the entity name

I selected a portion of the entity's signature, and searched for lexical matches in the new API using ECLIPSE's text-based search tool. The portion selected depended on the entity in question: for

classes, I selected the class name, and the **class** reserved word; for methods, the method name and its associated return type; and, for fields, the field name and its associated type. I found it necessary to add the **class** reserved word as searching without it often lead to an overwhelming number of mostly false hits.

5.1.2.7 Experience

For some cases, I was initially not able to find a replacement for the binary-incompatible entity using any of the previous six analysis techniques. In the course of later examinations of that API version, or a subsequent version of the same, I sometimes stumbled across functionality or code comments that seemed relevant to those prior unsolved cases, and eventually led me to an appropriate replacement. These cases serve to illustrate how difficult it was to "penetrate" the API [Robillard and DeLine, 2011], and as such I could only determine replacements by developing some degree of experience with the library and its change patterns.

5.1.3 Classifying Transformations

After each replacement was identified, I examined the differences between the binary-incompatible entity and the replacement functionality I had found, and attempted to classify the change that occurred. Initially, I used basic refactoring terminology (e.g., move, rename) to describe changes where appropriate, but as I progressed, this proved inadequate.

With more complex changes, I tried to succinctly describe each case, or mark it as unclassified. As I continued the examination of the API, its later versions, and the rest of the APIs in the study, I would often see a recurrence of an earlier unclassified transformation, and the new case would gave me ideas on what was common to both situations, eventually leading to a new descriptive classification for those changes.

In some cases, I determined that the change itself was a combination of other refactorings, or other change classifications I had noted, and recorded all those that I felt applied. Finally, I considered the effect of migrating client source code to accommodate this library change, and

recorded whether the resulting transformation would replace one entity reference in the original client code with one other (1:1), one reference with multiple references (1:n), multiple references with one (n:1), or multiple references with multiple, other references (n:m). In cases where the entity appeared to be deleted, I recorded it as a 1:0 transformation.

5.1.4 Verification of Recommendations

The data I have collected represents my best judgment of how to repair BIs introduced between each API version. With the exception of the tooling I used to detect the BIs, all of my work has been done either manually, or with the assistance of the standard tools available through ECLIPSE. While I have been as careful and thorough with my results as I could, there is a reasonable expectation that I have made some mistakes. To check my work, I undertook a small verification case study in which I selected a subset of the changes I identified to be reviewed by other developers. The analysis of the STRUTS 1.0.2 to 1.1.0 migration was the most difficult I encountered, so I chose it for verification.

I randomly selected 20% of the BIs in this particular STRUTS version transition (44 changes in total); I then selected 15% of those changes to be seeded as incorrect recommendations to ensure my participants did not simply agree with my results, but took the time to investigate each change and identified replacement.

To seed incorrect changes, I used changes detected in the empirical study where one or more techniques had made an erroneous recommendation. One was the ActionMapping.setActionClass(String) method, whose deprecation comment indicated a replacement that did not exist. Another was the ActionServlet.validate field that was used to check for old configuration formats, and was deleted to force migration to the new configuration format; this is also a lexical match to the ActionConfig.validate field, which is used to determine if a validation method should be used. The remaining 5 incorrect changes were randomly selected from the review set, and then using the recommendation techniques described in Section 5.1.2, I searched for reasonable but wrong replacements that participants would likely encounter.

I recruited two PhD students with industrial and relevant web-development experience to undertake the study. Each participant was provided with both versions of the library source code, and a spreadsheet listing the BIs selected for this verification study as well as the proposed replacement for each. The spreadsheet included the incorrect changes that I had seeded randomly. Each participant was asked to determine whether the replacement for each BI in the new API was correct or not, and to record their decision in the supplied spreadsheet; if the recommended replacement was wrong, I asked them to indicate what they believed the correct replacement was. If the participants were unsure whether a replacement was correct or incorrect, they were asked to record "unsure". Participants were allowed to use whatever tooling they wished to assist in their work; both used the standard tools available in the ECLIPSE IDE. The participants each took approximately 4 hours to review the supplied 44 changes.

Participant 1 (P1) agreed with 33 of the 37 correct changes (89.1%), and was unsure of 1; however, P1 also discovered one mistake I had made, finding a correct replacement for a method that was different than what I had recorded.[11] Participant 2 (P2) agreed with 35 of the correct changes (94.6%), and was unsure of 2. One of the changes that P2 agreed with was the one that P1 had determined to be incorrect. For the incorrect changes that I had seeded, P1 caught 4 of them, and missed 3, while P2 caught 3, was unsure of 2, and missed 2.

The three incorrect cases that P1 missed included both changes seeded from the erroneous recommendations, as well as a change where they felt a different replacement was correct. But had they checked the scope for their choice, they would have noticed it was inaccessible to the original entity and its subclasses. Participant P2 also missed both changes that were seeded from erroneous recommendations.

[11] I modified the corpus to repair this issue before using it to evaluate other approaches.

5.2 Observations and Results

I begin with an overview of the changes I observed in Section 5.2.1, followed by a discussion on the effectiveness of the change recommendation techniques I applied in Section 5.3, and ending with a description of characteristics and nature of the changes in Section 5.4. The detailed list of binary incompatible API changes, their replacements, form the Library Migration Corpus, which I have included as an appendix to this work (see Appendix E).

5.2.1 Overview of Changes

Table 5.1: Summary of BIs. + = public entities; # = protected entities.

		Class		Method		Field	
		+	#	+	#	+	#
STRUTS	1.0.2–1.1.0	76	0	53	28	1	73
	1.1.0–1.2.4	35	0	114	17	0	13
	1.2.4–1.2.6	0	0	0	2	0	14
	1.2.6–1.2.7	0	0	17	0	0	0
	1.2.7–1.2.8	0	0	0	0	0	0
	1.2.8–1.2.9	0	0	0	0	0	0
LOG4J	1.0.4–1.1.3	6	0	9	5	11	9
	1.1.3–1.2.1	11	0	87	8	1	6
	1.2.1–1.2.2	0	0	1	0	0	0
	1.2.4–1.2.5	2	0	1	0	0	0
	1.2.5–1.2.6	0	0	1	0	1	0
	1.2.7–1.2.8	0	0	0	0	0	0
	1.2.8–1.2.9	0	0	0	1	0	0
JDOM	1.0b6–1.0b7	2	0	8	5	0	7
	1.0b7–1.0b8	3	0	27	2	0	14
	1.0b8–1.0b9	1	0	16	4	0	5
totals		136	0	334	72	14	141

Table 5.1 presents an overview of the BIs I detected across all versions of the libraries I examined, with breakdowns based on the entity type (classes, methods, or fields), and visibility (public or protected).

No BIs were detected between versions of the DBCP and SLF4J APIs; this was initially

surprising, but I found that in the case of SLF4J its maintainers had made a special effort to preserve binary compatibility across all versions of their library.[12] Consequently, I have omitted those versions from the tables, although they are still part of the changeset corpus.

The complete dataset represents 697 binary incompatible changes across 16 API version transitions. Incompatibilities in methods accounted for the majority (58.2%) of all API changes, with class (19.5%) and field (22.2%) incompatibilities accounting for the rest. Most of the changes were made to public entities (69.4%), but nearly a third of all BIs were caused by protected entities (30.5%), two-thirds of which were fields.

5.3 Recommendation Techniques

Table 5.2: BIs resolved by a given number of techniques.

		0	1	2	3	4	5	6
STRUTS	1.0.2–1.1.0	31	130	51	15	3	1	0
	1.1.0–1.2.4	5	81	72	17	4	0	0
	1.2.4–1.2.6	0	14	2	0	0	0	0
	1.2.6–1.2.7	0	0	0	17	0	0	0
	1.2.7–1.2.8	0	0	0	0	0	0	0
	1.2.8–1.2.9	0	0	0	0	0	0	0
LOG4J	1.0.4–1.1.3	4	14	14	8	0	0	0
	1.1.3–1.2.1	5	8	82	18	0	0	0
	1.2.1–1.2.2	0	0	1	0	0	0	0
	1.2.4–1.2.5	3	0	0	0	0	0	0
	1.2.5–1.2.6	0	0	2	0	0	0	0
	1.2.7–1.2.8	0	0	0	0	0	0	0
	1.2.8–1.2.9	0	0	1	0	0	0	0
JDOM	1.0b6–1.0b7	2	9	8	3	0	0	0
	1.0b7–1.0b8	4	12	29	1	0	0	0
	1.0b8–1.0b9	4	5	15	2	0	0	0
totals		58	273	277	81	7	1	0

Table 5.2 presents an overview of how many of the analysis and recommendation techniques described in Section 5.1.2 (not including Experience) were successful in finding the right replacement

[12] www.slf4j.org/faq.html

functionality for each BI. Most BIs had only one (33.7%) or two (34.2%) analysis techniques that were successful in finding replacement functionality. Surprisingly, 21.2% of all BIs could not be resolved with *any* of the analysis techniques used; in these cases, I either needed to rely on my experience with the library to find the replacement, or I simply could not determine what happened to that functionality in the new library version. This may be an indication of how penetrable the APIs were [Robillard and DeLine, 2011].

Table 5.3 provides an individual breakdown of how often each of the analysis techniques described in Section 5.1.2 was successful in finding replacement functionality, how often it failed, and how often it provided us with an erroneous recommendation that had a high degree of confidence. Most techniques proved to be effective for roughly 20% of the changes examined, with little overlap (as indicated by Table 5.2). I also report how many cases were resolved due to experience here. The remainder of this subsection discusses how each technique fared.

5.3.1 Effectiveness of Call Analysis

The effectiveness of call analysis was highly variable, ranging from success rates of 9% in STRUTS, to around 40% for LOG4J and JDOM. Researchers have noted this technique relies heavily on APIs referencing their own functionality to glean change examples from [Dagenais and Robillard, 2011]; it would useful if measures of API self-referencing were devised that could be correlated with call analysis performance.

Call analysis was the least error-prone technique, aside from release documentation. Cases where call analysis led to errors could often be described as "code hypocrisy", where the public API was altered, but the original functionality was moved and kept for internal use. For example, tracing how references to the ActionServlet.application field in STRUTS 1.0.2 were replaced in version 1.1.0 leads to a recommendation that its replacement be an attribute stored in HttpServlet superclass, when it should be the Action.getResources(HttpServletRequest) method that is aware of the current module context, and returns a more appropriate attribute.

There were cases where call analysis fails to find replacements when portions of a call chain

Table 5.3: Effectiveness of techniques to detect resolutions. F = found; M = missing; E = error; CC = code comment; Ex = experience.

		original CC			release docs.			impl. details			call analysis			lexical search via: entity name			lexical search via: CC	Ex
		F	M	E	F	M	E	F	M	E	F	M	E	F	M	E		
STRUTS	1.0.2–1.1.0	15	211	5	30	201	0	120	111	0	24	206	1	59	158	14	9	37
	1.1.0–1.2.4	59	119	1	35	144	0	122	57	0	14	163	2	47	119	13	11	4
	1.2.4–1.2.6	0	16	0	0	16	0	16	0	0	2	14	0	0	16	0	0	0
	1.2.6–1.2.7	0	17	0	0	17	0	17	0	0	0	17	0	17	0	0	17	0
	1.2.7–1.2.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1.2.8–1.2.9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LOG4J	1.0.4–1.1.3	2	38	0	0	40	0	33	7	0	18	22	0	13	24	3	0	0
	1.1.3–1.2.1	48	53	12	61	52	0	64	49	0	37	76	0	14	97	2	2	0
	1.2.1–1.2.2	0	1	0	0	1	0	1	0	0	0	1	0	1	0	0	0	0
	1.2.4–1.2.5	0	3	0	0	3	0	0	3	0	0	3	0	0	3	0	0	0
	1.2.5–1.2.6	0	2	0	1	1	0	1	1	0	0	2	0	0	2	0	1	1
	1.2.7–1.2.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1.2.8–1.2.9	0	1	0	0	1	0	1	0	0	1	0	0	0	1	0	0	0
JDOM	1.0b6–1.0b7	2	20	0	1	21	0	18	4	0	10	12	0	3	19	0	0	0
	1.0b7–1.0b8	21	24	1	0	46	0	36	10	0	16	29	1	0	46	0	0	0
	1.0b8–1.0b9	4	22	0	0	26	0	20	6	0	15	11	0	2	24	0	0	0
totals		151	527	19	128	569	0	449	248	0	137	556	4	156	509	32	40	42

are also altered in the new API, such that a preceding caller also appeared to have been removed (e.g, the ActionServlet.destroyActions() method in Struts 1.0.2). Iterative approaches that reanalyze call chains when other changes are resolved could address this [Dig and Johnson, 2006b].

5.3.2 Effectiveness of Code Comments

BIs were rarely documented as deprecated, and only 21.7% of affected entities had replacement functionality described in their associated code comment. However, code comments proved to have the highest error rate of the analysis techniques I examined, with 10–20% of the documented replacements being incorrect. The vast majority of these cases occurred when replacement functionality was also removed in the new API, for example, as with the BasicConfigurator.disable(String) method in LOG4J 1.1.3, whose code comment recommended Category.getDefaultHierarchy().disable() as a replacement, but the Category.getDefaultHierarchy() method had disappeared in 1.2.1. The case described in Section 5.1.4 is another example of this phenomenon. Tool support to verify the existence of referenced entities in code comments would help API maintainers fix oversights like this, and would likely be straightforward to create.

5.3.3 Effectiveness of Release Documentation

Release documentation noting the introduction of binary incompatible changes between library versions varied heavily; I had a difficult time in finding release documentation for the relevant JDOM versions, and consequently the reported effectiveness here may be artificially low. Still, the majority of binary incompatible changes between API versions were not documented, with as few as 14.8% of all breaking changes documented in STRUTS. In contrast to code comments, I did not find any cases where release documentation was wrong about the replacement for a breaking change, and as such I only report the successes for this technique.

5.3.4 Effectiveness of Lexical Search (Via the Entity's Name)

Across all libraries, lexical searches on entity names found appropriate replacements for 22.4% of all BIs; however, performance for individual libraries varied significantly, as only 5.3% of breaking changes in JDOM had replacements locatable through lexical searches, as compared to 27.8% in STRUTS. This discrepancy is likely due to two factors: the degree of uniqueness in entity naming across API versions, and the percentage of changes in APIs that did not involve renaming or replacing functionality. I also found, unsurprisingly, that in cases where lexical searches return a single result with a high degree of similarity in the match, it was wrong 17.0% of the time; an example of this is presented in Section 5.1.4. In these cases, lexical search is easily confused when methods, fields, or classes with similar names represent very different functionality and behaviour. More sophisticated lexical search techniques for method matching exist than I have attempted here [Kim et al., 2007].

5.3.5 Effectiveness of Lexical Search (Via the Code Comment)

While my approach to searching on code comments is simple, there were several instances where this was the only technique that led to me to a replacement for a breaking change, such as for the ResetTag.getName() method in STRUTS 1.0.2, which was replaced with ResetTag.getProperty() in STRUTS 1.1.0. The method had no callers in Struts 1.0.2, was not deprecated, and had no associated release notes or code comments to indicate the that it was altered between versions. Further, the method had changed too significantly to be lexically matched, and had insufficient implementation details to be sure that the purpose of the method was for handling the same data as before. Only the similarity of the code comment between version helped to identify that these were equivalent.

5.3.6 Effectiveness of Implementation Details

The high success rate of this technique comes at a heavy price: examining and reasoning about implementation details to determine replacements for breaking changes was the most effort-intensive analysis I undertook. I think it is significant to note that for 35.5% of all BIs, I was unable to successfully determine what the replacement should be through a thorough, manual and tool-supported analysis of the source code.

5.3.7 Effectiveness of Experience

While there were few changes in most API transitions that required relying on my experience with the API to resolve, the STRUTS 1.0.2 to 1.1.0 migration was particularly problematic to deal with. A significant problem here was the poor quality of source code documentation: only 8.7% of the breaking changes had a code comment which indicated a replacement, and 25% of those code comments were wrong. I found that in subsequent releases of the STRUTS API, the overall quality of documentation improved with a greater number of BIs documented in the release notes, or in the entity's code comment, and thus it was far less necessary for me to rely on experience to resolve breaking changes.

5.3.8 RQ 5.1: How effective are recommendation techniques?

The examined analysis techniques proved, on average, to be ineffective for resolving the majority of API changes they were applied to. The most successful technique involved a detailed examination of the implementation details of the library source code by a human developer, and this only proved successful in resolving approximately two thirds of the binary incompatible changes; the success rates of the other techniques were far lower. Further, each binary incompatible API change was usually resolvable using only one or two of the considered change recommendation techniques: on average, a given recommendation or analysis technique was successful for only 20% of the BIs encountered.

5.4 API Transformations

In classifying how each API changed between versions, I borrowed terminology from the refactoring community to describe cases that clearly resembled meaning-preserving restructuring of source code [Fowler, 1999]; in others, I attempted to succinctly describe the transformation that occurred, and then clustered related changes under a more generalized term when appropriate. In cases where the transformations appeared to be a composite of other transformations, I applied multiple classifications to the change: 170 of the BIs I examined were such composite transformations. Consequently, I identified 1042 transformations that occurred across 697 BIs.

I have grouped these transformations according to the degree to which I believe that they can automatically enacted on client source code, as opposed to requiring some degree of specification or input from a developer on how to resolve semantic problems.

- Fully automatable transformations, described in Section 5.4.1.
- Partially automatable transformations, described in Section 5.4.2.
- Hard to automate transformations, described in Section 5.4.3.

5.4.1 Fully Automatable

Fully automatable transformations involve API changes that do not alter the semantics of the API significantly, and thus can be sufficiently reasoned about or captured by an API migration tool such that additional developer input is not needed to migrate dependencies on the binary incompatible entity to its replacement(s). I classified *Move*, *Rename*, *Pull Up*, *Remove Parameter*, *Parameters Reordered*, and *Encapsulate* transformations as being fully automatable. Descriptions of these transformations are readily available (e.g., in the work of Fowler [1999]), and I do not elaborate on them here.

I did not restrict using the *Encapsulate* classification to transformations only involving fields, as Fowler [1999] does, as there were clear cases where non-field functionality was being encapsu-

lated. For example, in LOG4J's JMSSink class, a substantial portion of the functionality was moved out of the main method body and into the class's constructor between versions 1.2.5 and 1.2.6, simplifying a series of configuration steps that client developers using the class would normally need to perform themselves. Because this sequence of calls was now hidden behind the abstraction of the class's constructor, I marked such changes an *Encapsulate* transformation.

5.4.2 Partially Automatable

Partially automatable transformations involve API changes that, in the general case, would be difficult to enact without additional specifications and input from a developer; however, in particular cases, entities have unique structural characteristics, or assumptions about their mode of use, that could allow migrations to be automatically enacted provided a series of preconditions are met.

5.4.2.1 Shorten Subtype Chain

This involves cases where the leaf subtype of an inheritance hierarchy was replaced with its immediate supertype. Mechanically enacting this change would normally require developers to determine how to remap methods, fields, and alternative behaviours that may have been present in the subtype that do not exist in its supertype; however, in cases like the ActionError class in STRUTS 1.1.0, some subtypes consisted entirely of constructor calls which were delegated to its supertype. While the general case cannot be automatically transformed, these cases could.

5.4.2.2 Encapsulate (Partial)

Encapsulate (Partial) refers to changes in which functionality was encapsulated, but its replacement did not provide the same set of functionality as previously existed. For example, the LoggingEvent.locationInfo field in LOG4J version 1.0.4 was changed from a **public** field to **private** in version 1.1.3, and a **public** getter method was provided, but not a setter. These transformations seem to occur when either

1. the API maintainers need to prevent access to functionality they should never have exposed in the first place; or,

2. the maintainers feel that a portion of the functionality is likely not being used (e.g., they assume client source code is not altering the contents of the locationInfo field).

Enacting this change on client code will not automatically work if the callers are using the encapsulated entity in ways the API maintainers need to prevent, or did not anticipate: in these cases, client developers will need to rework their code, perhaps significantly, to accommodate this change.

5.4.2.3 Expose Implementation

This involves several related transformations, which generally stemmed from a convenience method or class being removed or broken up, but the underlying functionality was still accessible to client code. Client callers were now expected to implement the exposed functionality directly, rather than rely on the library to provide convenience methods. For instance, in JDOM 1.0b7, the Document class provided several convenience methods, such as getProcessingInstruction(String target) which searched a content list for the first ProcessingInstruction found, and returned that to the caller. In the 1.0b8 API, callers were instructed to instead directly access the content list via getContent(), and perform the search themselves. In cases like this, client code would be migratable by simply inlining the API's previous implementation of the method (e.g., as in the work of Perkins [2005]).

In some cases though, the exposed implementation made several assumptions about the objects and data that the client code has access to in its calling context, which may not be true if the caller was relying on the API to gain access to them. For example, subclasses of FormBeanConfig in STRUTS 1.2.4 would need access to an HttpServletRequest object to obtain the data that in v1.1.0 was stored in the moduleConfig field. In other cases, the exposed implementation took the form of a new abstract class (e.g., ConditionalTagBase in STRUTS 1.1.0) which made it easier for the developer to provide their own specialized implementation of the missing functionality (e.g., IfAttributeExistsTag), but would require some insight to recognize the one or two key lines of code that need to be added to a new subclass to implement the new solution. The developer would need to provide input in these cases, before the migration could be enacted.

5.4.2.4 Consolidate

Consolidate consists of transformations in which several classes, methods, or fields were replaced by more generalized functionality; it is essentially the inverse of *Expose Implementation*. Problems arise though when the consolidated functionality includes additional operations or behaviours that may not have been expected by client callers, such as with the consolidation of the ActionServlet .initDebug() method in STRUTS 1.0.2 into the ActionServlet.initOther() method in 1.1.0. Because of this, I made a distinction between cases where functionality was simply *Encapsulated* in another form, but is otherwise the same (see Section 5.4.1), and where it was *Consolidated* but incorporates additional functionality which may have been part of the original calling sequence.

Some degree of restriction would be needed to ensure that replaced client code does not have new behaviours introduced which the client code did not expect.

5.4.3 Hard to Automate

Hard-to-automate transformations are cases that represent significant alterations to the semantics of entities between versions; a tool is unlikely to handle such a transformation without assistance from a developer (e.g., [Chow and Notkin, 1996; Balaban et al., 2005; Nita and Notkin, 2010]).

5.4.3.1 Replaced by External API

In these cases, the library removed functionality it previously provided in its API, and expected client callers to replace that functionality with equivalent types from other libraries. This introduces two problems.

1. The client developer must determine how to incorporate the external API if it does not ship with the new library version. This can happen in cases where the API adopts a facade to generically interact a variety of libraries (e.g., STRUTS adopted the APACHE COMMONS LOGGING facade to replace its internal logging implementation, requiring client developers to choose a logging library, and integrate it into its system so that the STRUTS facade can function).

2. The replacement is often designed with generic reuse in mind, and may be missing functionality that was specific to the previous API, or may require initialization steps that were previously taken care of by the API. Client developers will need to determine how to bridge this gap in their source code.

5.4.3.2 Add Contextual Data

This involves changes, typically to methods and to class constructors, in which additional information in the form of a new parameter was required in the new API. Unlike the "Add Parameter" refactoring [Fowler, 1999], these changes clearly expect that callers can gain access to the required data, which is non-trivial and cannot be substituted with a default value.

For example, code which called the Action.getResources() method to gain access to a MessageResources object in STRUTS version 1.1.0 were required in version 1.2.4 to supply an HttpServletRequest object as a new parameter. Some sort of dataflow analysis will be needed to see if the developer has access to that object type in their immediate calling context; if not, the developer needs to either gain access to that type, or excise this dependency from his code.

5.4.3.3 Change Type, Supertype Change, and Remap

The *Change Type*, *Supertype Change*, and *Remap* transformations are conceptually similar to each other. *Change Type* involves changes to a field's declaration type, a method's return type, or the type of one or more parameters in a method's signature. *Supertype Change* refers to classes whose supertypes changed, thereby altering the fields and methods they inherit. Finally, *Remap* involves systematic replacements of entities with other, similar entities. What is common to the three of these transformations is that functionality is being replaced wholesale, and it is not immediately clear that the functionality is completely equivalent.

A simple example of the problem this introduces can be found in the org.jdom.Comment class in jDOM, which merged the org.jdom.Comment:document field, whose type was Document, and the org.jdom.Comment:parent field, whose type was Element, into the org.jdom.Comment:parent field, but changed the type to Object, between versions 1.0.b7 and 1.0.b8. The motivation seems to have been

to save space: only one of these two fields could be set at a given time, depending on whether this was the root Comment node, or if it was a child node. By merging them, the library developer could use reflection to determine if this was the parent, or a child, and still provide the same getter/setter functionality as before.

However, if a developer had subclassed the org.jdom.Comment class to extend its functionality, and an API migration tool attempted to alter its usage of the org.jdom.Comment:parent field by casting to Element, a run-time exception will sometimes be thrown when the underlying object is actually a Document. The developer would need to describe the special case in which this field contains a Document object, and how to test for that.

5.4.3.4 Add to Interface

Here, new methods were added to an interface; thus, any client code which implemented those interfaces needed to supply implementations for several new methods. Dig and Johnson [2006a] notes that the Eclipse framework works around this problem by introducing numbered versions of interfaces, each of which extends the previous version, so that new functionality can be added without immediately breaking compatibility. However, this also means that developers can be confused by the differences between these interfaces, such as when they need to use IMarkerResolution instead of IMarkerResolution2 [Dig and Johnson, 2006a].

5.4.3.5 Extract to Configuration

This involves changes where entities were deleted because they would later be supplied in configuration files that could be loaded on application startup. This was particularly prevalent in cases where introspection was used to dynamically look up types or methods, such as in STRUTS's migration to JAVABEANS.

5.4.3.6 Immutable

Immutable describes changes that attempted to make an entity resistant to change, typically by adding the **final** modifier.

5.4.3.7 Hide

Hide describes cases in which the visibility of a method or field was reduced, thus restricting the access that callers have to that functionality. Unless this transformation is also accompanied by an *Encapsulate* transformation, the effect can be the same as deleting functionality from the API, particularly when the visibility is reduced to **private**.

5.4.3.8 Delete

These are cases in which I could not find any replacement for the old functionality in the new version, and thus were forced to conclude that no replacement existed, or the developers deliberately removed functionality to force migration to an alternative implementation.

Deletions were relatively rare; one case in the LOG4J library saw three classes deleted which were responsible for providing a help dialog option in a LOGGING graphical user interface that was deemed unnecessary.

5.4.3.9 Repair Deliberately Damaged Code

Repair Deliberately Damaged Code is a unique case, involving changes made to the Loader class in LOG4J in which a developer deliberately broke the signature of the loadClass method by changing its parameter type to a nonsensical value, preventing its use until a bug fix could be developed. This breaking change was part of public release, and was only repaired in 1.2.2.[13]

5.4.4 RQ 5.2: How feasible is it to automatically migrate dependencies?

Table 5.4 summarizes the kinds of transformations I observed between API versions, with respect to the number of entities being replaced and the number of replacement entities, as described in Section 5.1.2. The largest number of changes to client code are 1:n in nature (48.1%), with a nearly equal number being 1:1 (47.2%), and the rest comprising n:1 or n:m transformations, and deletions. Table 5.5 summarizes the number of BIs that I classified as involving a fully automat-

[13] http://issues.apache.org/bugzilla/show_bug.cgi?id=9305. I do not understand why the method was not simply commented out until the fix was applied.

Table 5.4: Transformation types.

		1:1	1:n	n:1	n:m	1:0
STRUTS	1.0.2–1.1.0	160	58	3	6	4
	1.1.0–1.2.4	71	108	0	0	0
	1.2.4–1.2.6	14	1	0	1	0
	1.2.6–1.2.7	0	17	0	0	0
	1.2.7–1.2.8	0	0	0	0	0
	1.2.8–1.2.9	0	0	0	0	0
LOG4J	1.0.4–1.1.3	20	19	0	0	1
	1.1.3–1.2.1	26	83	3	0	1
	1.2.1–1.2.2	0	1	0	0	0
	1.2.4–1.2.5	0	0	0	0	3
	1.2.5–1.2.6	0	2	0	0	0
	1.2.7–1.2.8	0	0	0	0	0
	1.2.8–1.2.9	0	1	0	0	0
JDOM	1.0b6–1.0b7	7	10	1	2	2
	1.0b7–1.0b8	12	32	0	0	2
	1.0b8–1.0b9	19	3	0	0	4
totals		329	335	7	9	17

able, partially automatable, or hard-to-automate transformation across all three libraries and their versions. In cases where I determined that multiple transformations had been applied, I rated the complexity of the transformation as that of the most difficult transformation assigned to it.

Nearly two-thirds of the needed migration transformations are hard-to-automate: defining how to migrate developer source code for these transformations will require some form of additional specification by a developer. The nature of this additional specification will vary, depending on the cause of the change: if an API type has changed, a specification of how its replacement is equivalent (e.g., defining how functionality from a removed class is equivalent to its identified replacement) would be required, while **interface** additions or the hiding of functionality may require a developer to provide additional implementation work in their software.

5.5 Discussion

Three issues remain to be discussed.

Table 5.5: Automatic transformability of BIs. F = fully automatable; P = partially automatable; H = hard to automate.

		F	P	H
STRUTS	1.0.2–1.1.0	103	19	109
	1.1.0–1.2.4	23	10	146
	1.2.4–1.2.6	14	0	2
	1.2.6–1.2.7	0	0	17
	1.2.7–1.2.8	0	0	0
	1.2.8–1.2.9	0	0	0
LOG4J	1.0.4–1.1.3	9	4	27
	1.1.3–1.2.1	13	4	96
	1.2.1–1.2.2	0	0	1
	1.2.4–1.2.5	0	0	3
	1.2.5–1.2.6	0	2	0
	1.2.7–1.2.8	0	0	0
	1.2.8–1.2.9	0	0	1
JDOM	1.0b6–1.0b7	7	1	14
	1.0b7–1.0b8	10	8	28
	1.0b8–1.0b9	10	0	16
totals		189	48	460

5.5.1 Recommendation Blindness

In the verification study in Section 5.1.4, I described the verification study I performed to check the correctness of my work. I noted that two changes were seeded as errors in the study, to check the quality of my participant's work were both missed in their results. The errors came from actual false-positives encountered while investigating the library changes, using the inference techniques discussed in Section 5.1.2: one came from an erroneous code comment which referenced replacement functionality that did not exist; the other was a false-positive match obtained through lexical analysis which superficially appears to be correct.

It is significant that both participants failed to detect wrong change recommendations in cases where a tool/technique had strongly recommended a replacement. I feel this is unlikely to be caused by the participants simply not having examined the code, as a second false change I seeded, which also pointed to nonexistent functionality, was detected by both participants. I also do not

believe that this is due to an inability by the participants to reason about the behaviour and intent of code, as both candidates caught other mistakes with these characteristics, and one participant was able to point out a case in which I was wrong about a replacement. In fact, when it was pointed out to participant P1 after the study that they had missed the case of the erroneous code comment, they were surprised: they remembered checking the replacement, noting:

> "... I guess I was just looking at the code in that area, I wasn't really looking for that [specific] method."

I believe that the participants experienced "recommendation blindness" in their work: a precise and unambiguous recommendation that superficially seemed reasonable was implicitly trusted. This recommendation blindness caused the developer to ignore clear evidence in the source code, that would normally have caused them to correctly conclude the recommendation was wrong.

I feel this has important implications for recommendation tools in two respects:

1. the consequences of false predictions are serious, especially when the tool or technique provides a high degree of confidence about the recommendation;[14] and,

2. researchers who analyze only the predictions of their tools as a means of estimating their accuracy, such as those described in Section 3.4.2, are at risk of being biased by their tool's predictions.

5.5.2 Discrepancies with the Literature

There are discrepancies between the numbers I report in the results of Section 5.4, and those reported in the study of library evolution by Dig and Johnson [2006b], as well as the evaluations of both Schäfer et al. [2008] and Wu et al. [2010] of their proposed techniques, with respect to the changes discovered and classified in the STRUTS 1.1.0 to 1.2.4 migration.

[14] A replacement description in a code comment should be, in my opinion, a recommendation in which we can be highly confident.

5.5.2.1 Comparison with Dig and Johnson

In the work of Dig and Johnson [2006b], a different number of changes are reported as having occurred between the 1.1.0 and 1.2.4 version of STRUTS. This difference is easily explained by my choice to identify changes based on the mechanical determination of where BIs exist between two API versions, and the choice to include all API entities, including protected entities, in the analysis. Dig and Johnson restricted their study to public entities they deemed intended for reuse, and based their analysis on published release notes as well as a heuristic analysis of the source code. The methodologies for determine breaking changes are very different, but the methodology I have chosen is much broader in scope, while also having the advantage of being easier to replicate.

Dig and Johnson also reported that, of the breaking changes they discovered in STRUTS 1.1.0 to 1.2.4, 90% were "refactorable" [Dig and Johnson, 2006b, p. 103]. In contrast, I found that only 12.8% of the BIs in that particular migration were easily performed by mechanical transformation tools; the majority of the changes either require restricted conditions to be met, or some degree of additional specification from a developer that could not be automatically inferred by a tool.

To a large degree, this discrepancy is due to how Dig and Johnson classify changes: they are intent on capturing refactorings that alter structure while preserving behaviour, and classify several serious breaking changes (such as deleted class, deleted method, change argument type, changed return type, pushed down method) as being refactorings. While I encountered similar changes, I do not believe that these changes would be easily handled by mechanical transformation tools; instead, the API maintainer, or the client developer, would need to craft some minimal specification that would describe how to remap classes to accommodate these breaking changes.

I note that Dig and Johnson also acknowledged this issue in their work, pointing out in the case of "change return type" that some sort of additional mapping would need to be provided to describe how the new return type mapped to the old [Dig and Johnson, 2006b, p. 96]. Thus, while some of these changes may be caused by refactorings, or are refactorable, many require some degree of additional input by one or more developers, before an automated approach could attempt to redress

the breaking changes.

5.5.2.2 Comparison with Schäfer et al.

The work of Schäfer et al. [2008] was also evaluated on the STRUTS 1.1.0 to 1.2.4 migration, and reported a different number of API changes. At the suggestion of an anonymous reviewer of my work, I manually applied the call analysis technique of Schäfer et al. [2008] to the STRUTS 1.1.0 to 1.2.4 API migrations to replicate their work, and understand the nature of the discrepancy.

As discussed in Section 3.3.3, Schäfer et al. [2008] applies call analysis, but also uses any available instantiations of the library, such as example applications, test suite code, or other systems which have already been migrated to a new API version, to glean examples of how to migrate dependent code in response to breaking changes in the API. In the evaluation of their approach, Schäfer et al. indicate that a key assumption in their work is that:

> "... (A1) Users of the changed framework *exist* [emphasis added] that are ported to the new version; ..." [Schäfer et al., 2008, p. 477].

They justify this assumption in part by their evaluation of the STRUTS 1.1.0 to 1.2.4 migration, claiming that:

> "Assumption (A1) is realistic for large open-source frameworks. There are often early adopters of the changed framework with detailed knowledge of the framework code. For instance, when the Eclipse platform changes, the Java Development Toolkit is one of the first instantiations that is ported. Further, *the Struts experiment has shown that our approach performs reasonably well, even if no instantiations are available beyond test cases* [emphasis added]. The latter are available for a large number of frameworks and they are typically ported to new versions because they are maintained by the framework developers" [Schäfer et al., 2008, p. 477].

To replicate their work, I sought test cases for, or references to, the relevant entities in the test suite associated with Struts 1.1.0, and then manually compared their usage of the API to their later usage in the 1.2.4 version of the test suite, to see if their usage of any of the 179 BIs present in that migration had been migrated in the test suite code.

I found only two cases where a BI had an associated test case, or was used in another test case, and was migrated to a replacement in the new version of the test suite; both of the migrations matched the changes I had determined in the study. I found a third case in which a BI had already been migrated in the test suite for 1.1.0, in preparation for the eventual change in the API. Since their evaluation was conducted on a single library migration, they would have not captured this change.

Of the remaining BIs, their representation in the test suites was either happenstance, or completely absent. 21 field or method BIs had a unit test associated with their enclosing type, but none of the test cases exercised their functionality. 28 field or method BIs were contained in an enclosing type which was referenced in another test case, but again their functionality was not exercised. The remaining 127 BIs in that migration had no associated test case at all, nor were their enclosing types referenced in any other test case.

These results invalidate the assumption of Schäfer et al.: the test suite associated with a library may not provide sufficient API migration examples to be of use. The test suite provided with STRUTS 1.1.0 addressed barely more than 1% of the BIs caused by the API changes. While Schäfer et al. identified 66 breaking changes and their replacements, at most two of those could have involved BIs changes for which test code provided a migration example. The other transformations involve BI API changes gleaned from the library source code, non-BI API changes gleaned from the test suite or the library's source code, or are false positives, in which client or test suite code changed for reasons other than API migration.

Schäfer et al. [2008] are not clear in their presentation of their work as to whether or not the library source code is also used by their tool to glean usage examples of API migration. The paper

seems to suggest that it does not, and relies only on the usage of test suite code:

> "On the other hand, the Struts experiment covers the situation where there is no access to ported instantiations; instead, we used test cases of the project as the learning basis (379 classes)" [Schäfer et al., 2008, p. 476].

And later:

> "... The reason for such false negatives was that the analyzed instantiations did not use the respective parts of the framework...As none of the analyzed instantiations used the validation part of Struts, we did not find usage changes for these refactorings. For such cases, it seems beneficial to combine our analysis with existing techniques that *analyze the framework code* [emphasis added], as discussed in Section 7" [Schäfer et al., 2008, p. 478].

However, in his Master's thesis, Jonas reports the same results, included a detailed breakdown of the changes found, but is clear that the library's source code is part of the analysis, not just the associate test suites [Jonas, 2007]. This is a more plausible explanation of how [Schäfer et al., 2008] were able to detect 66 changes to the API, when so few of the breaking changes were covered by test suite code.[15]

5.5.2.3 Comparison with Wu et al., and with Meng et al.

In their evaluation of the AURA tool, Wu et al. [2010] note that a feature of their approach is in their tool's ability to detect when functionality (specifically, methods) were deleted between versions of an API, as a means of improving tool accuracy. They presented two evaluations where

[15] In my original publication [Cossette and Walker, 2012], the authors had not responded to my inquiries, leaving me to assume (for the reasons listed above) that Schäfer et al. had only analyzed the test suite code of the STRUTS framework. After reviewing the thesis of Jonas [2007], I feel that the authors were accidentally unclear in their conference publication [Schäfer et al., 2008], and have rectified my analysis accordingly.

the effect of accounting for method deletion was compared [Wu et al., 2010, pp. 331, 333]; their investigation found that 77 methods were deleted between STRUTS 1.1.0 to 1.2.4 [Wu et al., 2010, p. 332]. When AURA accounted for method deletion, it raised the relative recall of their tool by 55.9% when compared to prior work. When deleted methods were not accounted for, their approach had lower recall than prior work. The same approach is used by the subsequent HIMA tool of Meng et al. [2012].

However, I found no cases where an API entity was deleted without an appropriate replacement between STRUTS 1.1.0 and 1.2.4. Finding replacements in this transition is difficult, requiring systematic investigation and extensive reasoning. While some API replacements involved functionality external to the STRUTS API, many of the API changes marked as deleted by AURA or HIMA had replacement functionality present within the STRUTS API (see Appendix D). This highlights the need for a complete and validated set of change data for multiple libraries that undergoes continuous improvement, so that it is clear what changes any novel technique does and does not address.

5.5.3 Threats to Validity

The most significant threat to my work is internal validity: the empirical study relies heavily on the correctness of the replacements I discovered for deleted functionality, the appropriateness of the classifications I applied to each change, and the consistency with which I applied the various recommendation techniques. The verification study provides some evidence that my work is correct, with only one mistake found in a sample of 20% of the data from the most challenging library migration to analyze.

I acknowledge that errors may remain in this data; however, the manual nature of the analysis is unavoidable, a fact also acknowledged by Dig and Johnson [2006a] in their empirical work. The change recommendation techniques I applied were not able to find replacement functionality for a BI change in nearly 80% of the cases examined, suggesting that there is currently no single tool or technique which is sufficiently reliable to build such a dataset. Furthermore, there is clearly a need

to know how libraries have evolved through investigation that is independent of tools' reported solutions (see Section 5.5.1); my manual investigation calls into question the reported results of other state-of-the-art tools.

A related issue is whether or not the corpus created through manual investigation can be used to calculate the false negatives present in the analysis of the above techniques. While I have reduced the problem of identifying API changes to that of identifying binary incompatible changes, allowing for a mechanical determination of what has changed in these APIs, the problem of identifying equivalent functionality which should be used to redress the impact of the API change is an undecidable problem. Ideally, test suites would exist or could be created that would capture each BI, and prove that the replacement functionality I determined does in fact work; such a suite would provide tremendous confidence in the correctness of my results. Sadly, few (if any) test cases existed in the studied libraries that had direct bearing on the BIs I found, as noted in Section 5.5.2.2, meaning that all of these test suites would need to be created. Since the majority of the changes I found were classified as "hard to automate", automatically constructing such a test suite would also be difficult. Manually creating test suites to verify changes would be necessary, but also restricted in their coverage of the problem: many of problems posed by "hard to automate" changes are require the client developer to make decisions about what the appropriate adaptation for that change is in their calling context. A single test case could not represent the different impacts the same change could have on other, diverse software systems.

However, I disagree that the difficulty of precisely identifying false negatives means that this corpus cannot be used for this task.

1. Because the list of API changes are mechanically determined, it is possible to determine the upper limit on how many false negatives may exist. In cases where tools fail to make a recommendation for a binary incompatible API change, it follows that these are false negatives. As techniques are now attempting to identify cases where functionality is removed without replacement (e.g., [Wu et al., 2010; Meng

et al., 2012]), we can still rightly consider a failure to make recommendations in cases where functionality was in fact removed to still be a false negative.[16]

2. In cases where discrepancies exist, detailed reasoning or working examples should suffice to resolve disputes as to correctness. In Appendix D, I provide a thorough comparison of the results of the Corpus to the work of Dig and Johnson [2006a], Schäfer et al. [2008], and Wu et al. [2010], detailing cases in which there is agreement as well as disagreement, and establishing whether disagreements are due to a mistake in the Corpus, an error by the compared technique, or the reporting of a non-BI change. There are no cases in which the Corpus was found to be incorrect; however, a substantial portion of the correct recommendations given by other techniques concern non-BI changes to the library or API, and some techniques show significant rates of error that were not previously reported.

My work also suffers from issues of external validity: the STRUTS, LOG4J, JDOM, DBCP, and SLF4J APIs are not necessarily representative of all JAVA libraries, let alone all software libraries in active use. My goal in this work was to provide an in-depth study of these libraries, to suggest the possible transformations that all libraries may go through, and to provide some empirical comparison as to how well various change recommendation tools work. However, the scalability of studies like this is currently problematic, due to the necessary reliance on manual investigation.

5.6 Summary: Empirical Study of Reactive Change

Library migration, and how to redress the impact of changes in APIs, remains a challenging proposition. To better understand its nature, I have conducted a detailed, largely manual investigation of

[16] Wu et al. [2010] take this approach with their AURA tool, as they compare its detection of deleted functionality to techniques that make no such determination; correct detections are marked as a precision and recall improvement over other approaches (see Section 5.5.2.3).

the binary incompatibilities between pairs of library versions, to determine what adaptive changes to client code would be needed to migrate to subsequent versions. In addition, I have evaluated a range of change recommendation techniques for their ability to recommend the correct changes.

I found that for any given BI, it was rare that more than one or two change recommendation techniques was able to identify the correct replacement: on average, a given recommendation or analysis technique was successful for only 20% of the BIs encountered. I suggest that there may not be a single change-recommendation approach that is generally suitable for all situations; rather, hybrid techniques which incorporate multiple, complementary analysis techniques may prove more effective overall. Recent work has begun to try such approaches [Wu et al., 2010], but the work here questions those results.

For library migration research, the results are sobering: the majority of API changes I observed could not be automatically transformed in the client code without either severe restrictions on the contexts in which the transformation could be attempted, or some additional specification work by an API maintainer or consumer. This may point to limits on how effective migration techniques based on automatically detecting or capturing refactorings can be. That said, I do not feel that this is a negative result for API migration: many transformations *could* be semi-automatically enacted, if some additional information (such as how a previous type's methods map to its replacement) were provided by a developer. Tools that supplement automated library migration techniques with developer provided specifications or feedback may prove more effective, and easier to develop, than fully automated approaches.

Chapter 6

Using Structural Analysis for Reactive Change

With a ground truth understanding of how software libraries and their APIs change (see Chapter 5), it is apparent that the available analysis techniques for API change recommendation are insufficient in resolving binary incompatible changes to APIs. However, such support is necessary: the libraries studied frequently introduced serious incompatibilities between API versions, and most library's changes were not documented. Client developers of the library are thus often provided with no guidance on how to repair broken functionality. New inference techniques that can complement the gaps in existing approaches may be of use in addressing these shortcomings.

In this chapter, I present a new analysis technique for API change recommendation—*matching via structural generalization* (MSG)—to discover replacement functionality for breaking changes to APIs. MSG measures the structural similarity (and, to a lesser degree, the semantic similarity) between the implementations backing the API versions. It determines where structural similarities exist between entities from the old and new APIs, by greedily applying a similarity metric to determine which entity in the new API has the most structure in common with each entity in the old API. MSG has been reified in a prototype tool called UMAMI, which was evaluated by using change data I had previously established from four different JAVA libraries, comprising 16 version transitions. My evaluation shows the approach offers significant improvements over several common analysis techniques in this area, and can complement their predictions effectively.

The rest of this chapter is organized as follows: in Section 6.2, I introduce the MSG approach and detail how it works. I then present the research prototype tool UMAMI, which applies MSG to API change recommendation, in Section 6.4. The evaluation of UMAMI is explained in Section 6.5, and I compare the results of the UMAMI tool to other API change recommenders in Section 6.5.3. I discuss the implications of the results in Section 6.6.

AST_1: **variable_declaration**(
type(“int”),
variable_declarator(“i”),
initializer(**numeric_expression**(**expression**(“1”), “+”, **expression**(“2”)))
)

AST_2: **variable_declaration**(
type(“int”),
variable_declarator(“j”),
initializer(**numeric_expression**(**expression**(“3”), “+”, **expression**(“2”)))
)

AST_{AU}: **variable_declaration**(
type(“int”),
variable_declarator(Y),
initializer(**numeric_expression**(**expression**(X), “+”, **expression**(“2”)))
)

Figure 6.1: Example ASTs (AST_1 and AST_2) and a possible anti-unifier (AST_{AU}).

6.1 Background: Anti-Unification

The ideas behind *Matching via Structural Generalization*, and UMAMI, are based on *anti-unification*. For the sake of a self-contained presentation, I provide background information on anti-unification here; the presentation of my work continues in Section 6.2.

Anti-unification is a formal model that seeks to determine commonalities and differences between pairs of structures (in our context, structures are labelled trees, specifically abstract syntax trees); in doing so, it matches elements in one structure to elements in the other structure. Commonalities between the structures consist of the matched elements (nodes in the tree) that are identical; differences consist of the matched elements that are not identical. In anti-unifying two structures, a third structure is created called an *anti-unifier*, that represents these commonalities and differences, i.e., it represents a generalization of the two structures. Common elements are represented in the anti-unifier by a copy; differences are represented by *structural variables*. Substituting the structural variables in an anti-unifier with the appropriate original label, allows one of the two original structures to be recovered.

For convenience, I use the notation $v_1(v_2, v_3, ..., v_n)$ to denote a tree with a root node labelled v_1 possessing $n - 1$ child nodes with the indicated order. For example, Figure 6.1 illustrates the ASTs obtained from the code snippets **int** i = 1 + 2 and **int** j = 3 + 2 and a possible anti-unifier for them in which X and Y are structural variables. Note that this anti-unifier (roughly, **int** Y = X + 2) is not unique, as variable_declaration(X) and even simply X are also anti-unifiers for this pair. The anti-unifier given in Figure 6.1 is more specific than these, maintaining all the commonality of the original two structures, and so it is preferred. Unfortunately, a unique, most specific anti-unifier does not exist for most variations of anti-unification in the general case [Wagner, 2002], a challenge for any practical application.

The simplest form of anti-unification (syntactical or first-order anti-unification [Plotkin, 1970]) can be performed via a pre-order traversal of the two trees [Huet, 1976]: starting at the two roots, each encountered node is compared; if they are identical, a copy of the node can be added to the anti-unifier; if they are not identical, a node labelled with a *structural variable* is added to the anti-unifier representing the subtree rooted at each of the current nodes, and the children of the current nodes are not visited. For example, the trees $+(1, 2)$ and $+(3, 2)$ would anti-unify to obtain $+(X, 2)$, where X is a structural variable. The trees $+(1, 2)$ and $-(1, 2)$ would unfortunately anti-unify to obtain X, despite the common details in the leaves. The trees $+(1, 2)$ and $+(2, 1)$ would anti-unify to $+(X, Y)$ as the order of nodes is strictly maintained.

Higher-order anti-unification [Hasker, 1995] addresses these limitations by adding the ability to insert structural variables of higher-order (hence the name) in place of any node, and not just leaves or entire subtrees. Thus, $+(1, 2)$ and $-(1, 2)$ can anti-unify to $X(1, 2)$.

More general still is E-generalization [Burghardt, 2005] (or higher-order anti-unification modulo theories [Cottrell et al., 2008]), in which nodes can be allowed to match when not identical if they conform to properties defined in an *equational theory*. Such an equational theory can, for example, include equations that define a neutral element for a particular operator, allowing the creation of nodes out of nothing for the purposes of generalizing otherwise dissimilar structures.[1]

[1]The ability to arbitrarily generate structures from nothing poses quite a potential challenge for this application; as

An example of the usefulness of E-generalization can be seen in dealing with a programming language like JAVA, where while-loops and for-loops have different syntax for what are effectively identical concepts. Consider the code snippets **while** (i > 0) and **for** (; ; i > 0) whose trees can be represented with while$(>(\mathrm{i}, 0))$ and for$(\mathrm{NIL}, \mathrm{NIL}, >(\mathrm{i}, 0))$. These trees cannot be anti-unified using only higher-order, as the **while** statement has only one argument, while the **for** statement has three. Instead, we can define an equational theory describing how **for**-loops to **while**-loops are equivalent, such as $while(X) = for(\mathrm{NIL}, \mathrm{NIL}, X)$, and using this theory we construct a generalization of the two that preserves greater commonality between their structures, possibly representing them as $\mathrm{loop}(>(\mathrm{i}, 0))$.

There are issues with all these variations. Attempting to pick an arbitrary most specific anti-unifier remains problematic, as E-generalization is generally undecidable [Burghardt, 2005]. In this context, the structures and equational theories may also not contain the needed (semantic or conceptual) information to find generalizations with greater commonality.

6.1.1 Applications of anti-unification

Oancea et al. [2005] presents a single-pass, syntactic anti-unification algorithm to generalize algebraic equations to identify common properties. Cottrell et al. [2007] investigated the application of syntactical anti-unification on how two classes can be generalized as a refactoring target. Bulychev and Minea [2008, 2009] present an approach for detecting duplicate code using syntactical anti-unification to calculate the edit distance (the number of substitutions) between two code fragments at the statement level. These approaches require the structures to be near identical to be effective and can be thrown off by syntactic changes as simple as a change in statement orderings.

To address the limitations of syntactical anti-unification, Cottrell et al. [2008] investigated the application of E-generalization, allowing commonalities embedded within much greater structural variation to be handled. In particular, they use basic semantic knowledge of a programming language in generalizing structures with divergent syntax (e.g., **while**- and **for**-loops, as presented

such, an additional requirement is made that such structures are not allowed to appear in the anti-unifier.

in Section 6.1). Their approach, embedded in a tool called JIGSAW, was able to extract the detailed commonalities and differences between two code fragments and their environments such that code from one fragment can be semi-automatically injected and modified to fit into the other for small scale code reuse, based on the connections established by anti-unification of the two environments. This technique could be useful in helping to find replacement functionality for breaking changes as it:

1. is not thrown off by lexical and syntactic changes in the code when other implementation details can point to two entities being similar (a problem common to tree differencing approaches, e.g. [Fluri et al., 2007]);

2. can compare nodes in any location within an AST, regardless of how they are nested; and,

3. is able to consider differing orders of code features in determining whether two structures are equivalent, by introducing an equational theory that makes particular orders irrelevant..

6.2 Matching via Structural Generalization

The goal is to recommend replacement functionality for developers whose software relies on entities removed from an API in a new library version. Ideally, I want to determine matches (also called correspondences) between the elements of successive API versions: dependencies on entities in the old API version should be changed to the corresponding entities in the new API version, where these are not otherwise backwards-compatible. I assume that, in most cases, functionality that disappears from a particular API is actually still present in some form in the new API [Godfrey and Zou, 2005]: e.g., the functionality may be moved or renamed, replaced by alternative yet equivalent entities, or incorporated into existing API functionality (see Section 5.2). Because the functionality is "missing" from the API, I expect that some lexical or syntactic changes have

occurred, and it is possible that the underlying implementation is also changed. But, if equivalent functionality does exist in the new API version, it will likely have similar capabilities to the missing functionality, and should have some degree of similarity between their implementations. Further, I would like the approach to:

1. not rely on special-case library attributes;
2. avoid expensive semantic analyses; and,
3. correctly recommend matches between API versions.

Matching via structural generalization (MSG) applies the anti-unification techniques presented in Section 6.1 to determine correspondences based on the syntactic structures of the library implementations underlying the API versions. Because software systems are constructed using languages that have a well-defined syntax and semantics, it is possible to extract well-formed structures—e.g., abstract syntax trees (ASTs)—that describe API artifacts, such as method declarations, and then systematically compare their structural representation to other methods by identifying common substructures between them (equivalent structures are also known as *correspondences*).

Determining a method's most likely replacement in the API migration relies on finding which pair of methods share the highest structural similarity through the largest number of corresponding substructures (e.g., statements, logical tests, for-loops, etc.). In this section, I present the approach in two stages: I outline the computation of similarities between nodes in the ASTs in order to determine potential matches (Section 6.2.1), and I describe the selection of a subset of the potential matches to constitute the overall match (Section 6.2.2).

6.2.1 Computing Similarity between Nodes

To determine the similarity of two nodes, I have reused a similarity function from Cottrell et al. [2008] which was originally developed to assist in the small-scale reuse of code. The function relies on syntactic similarity, augmented with simple knowledge of semantic equivalences supported by

differing syntax, e.g., for-loops and while-loops in JAVA. This function depends on the types of the nodes being compared; syntactically identical leaf nodes possess similarity 1, and nodes of differing type generally have similarity 0. Special cases arise in which the semantics of the language permit the comparison of nodes of differing type, meaning that nodes of differing type can sometimes have similarity >0.

In addition, various heuristics are used in the similarity function. As a simple example, consider comparing the JAVA keywords **int** and **long**: these are not identical, so a similarity of 1 would be inappropriate, but they are not completely unrelated (an **int**-typed value can be assigned to a **long**-typed variable, for example), so a similarity of 0 would also be inappropriate; the arbitrary value 0.5 is used as a compromise. As a second example, simple names (identifiers) are compared using a normalized computation of the longest common substring, for simplicity, as alternative string comparison algorithms would be more complex to implement and debug and more expensive to compute without obvious a priori improvements arguing for their use. In the following algorithms, this function is referenced as SIMILARITY(χ_A, χ_B), where $\chi_A \in n_A, \chi_B \in n_B$, and n_A and n_B are the sets of nodes comprising the two trees being compared.

Inner nodes are compared recursively as shown in INNER-NODES-SIMILARITY. The similarities of child nodes are computed exhaustively in a pairwise manner. For each child node[2] of the inner node in one subtree, its highest similarity with any child node of the inner node in the other subtree is retained; all these similarities are summed and normalized by the size of the largest of the subtrees.

Note that any function $f(a, b)$ that returns a value in $[0, 1]$ (and for which $f(a, b) = f(b, a)$ and $f(a, a) = 1$) can be defined as a similarity function, and that there is no inherently "correct" similarity. Thus, the utility of a given similarity function can only be determined relative to its ability to calculate the results that are expected for a particular application.

[2] I denote a node and the tree rooted by that node, in identical manner.

```
INNER-NODES-SIMILARITY(n_A, n_B)
1  sum ← 0
2  foreach χ_A ∈ n_A
3       do max ← 0
4          foreach χ_B ∈ n_B
5               do σ[χ_A, χ_B] ← SIMILARITY(χ_A, χ_B)
6                  if σ[χ_A, χ_B] > max
7                    then max ← σ[χ_A, χ_B]
8          sum ← sum + max
9  return sum / max{|n_A|, |n_B|}
```

6.2.2 Determining Correspondence

I use the similarity function in determining which nodes correspond best between the two ASTs. For a gentle introduction, I begin with a naive version of the algorithm (NAIVE-MSG), which compares all pairs of nodes in the ASTs of the API versions. It calculates the structural similarity of all these pairs, and obtains a globally-optimal set of matches between the nodes of the ASTs.

This naive description would be prohibitively expensive to compute; thus, simplifying restrictions and approximations are used.

1. Rather than attempt to find an optimal set of correspondences, a greedy algorithm is used to repeatedly select the current globally-best match, removing the nodes involved from further consideration.

2. A threshold eliminates any potential matches below a minimum similarity.

3. It is assumed that node matches will usually be found within a class with the identical name in the AST for the new library version; only if no such potential match is found that satisfies the thresholds is the search broadened.

The MSG procedure takes two ASTs and attempts to match nodes in one against nodes in the other. An augmented AST (called a *correspondence AST* or CAST), which is initialized to an empty structure in line 1, has added to it the results of individually attempted matchings of pairs of

NAIVE-MSG(A, B)

```
1  foreach n_A ∈ A
2      do foreach n_B ∈ B
3          do σ[n_A][n_B] ← SIMILARITY(n_A, n_B)
4  foreach M ⊆ A × B
5      do σ_total[M] ← 0
6         foreach (n_A, n_B) ∈ M
7             do σ_total[M] ← σ_total[M] + σ[n_A][n_B]
8         σ_total[M] ← σ_total[M]/|M|
9  return arg max_M (σ_total[M])
```

unmatched methods (lines 2–4). A final call to FIND-BEST-FIT winnows the potential matches to the "best" ones, returning these (line 5).

The FIND-MATCHES procedure takes two ASTs, traversing the nodes therein to record all potential matches above the similarity threshold within a newly created CAST structure. The FIND-COMPARABLE-NODES procedure simply traverses the second AST looking for nodes of the same type as χ_A or of semantically comparable types (e.g., for-loops when χ_A is a while-loop); the set of these are returned. Any potential matches above the similarity threshold are added to the CAST as a correspondence edge (χ_A, pm) annotated with the computed similarity. As a result, the CAST contains both original ASTs plus a set of correspondence edges; each node might possess zero correspondence edges, one correspondence edge, or more than one. Nodes with zero correspondence edges are unmatched; nodes with more than one correspondence edge require winnowing at a later stage by FIND-BEST-FIT.

MSG(A, B)

```
1  cast ← ∅
2  foreach m_A ∈ methods[A]
3      do foreach m_B ∈ methods[B]
4          do cast ← cast ∪ FIND-MATCHES(m_A, m_B)
5  return FIND-BEST-FIT(cast)
```

```
FIND-MATCHES(A, B)
1  cast ← CREATE-CAST(A, B)
2  foreach χ_A ∈ A
3       do PM ← FIND-COMPARABLE-NODES(χ_A, B)
4          foreach pm ∈ PM
5               do sim ← SIMILARITY(χ_A, pm)
6                  if sim > τ_similarity
7                    then ADD(cast, χ_A, pm, sim)
8  return cast
```

The FIND-BEST-FIT procedure takes a CAST and greedily selects the most similar potential matches as the "best" ones. The potential matches are globally sorted in descending order of similarity (lines 1–5). The potential match with highest similarity is selected (line 7) and recorded as the "best" match for the two nodes involved (line 8). All potential matches of those two nodes are deleted (lines 8–18). The process continues until all potential matches have been eliminated from the CAST, at which point, it is returned (line 19).

Any nodes that still have no matches are assumed to actually possess no matches, i.e., an API method has been eliminated with no detectable replacement present in the new API version. Note that, to simplify the technique, I focused on API changes involving single method migrations (i.e., 1:1 changes) [Dagenais and Robillard, 2011; Schäfer et al., 2008], but this is not a fundamental limitation of the approach.

6.3 Is MSG Appropriate for API Change Recommendation?

Matching via structural generalization, and anti-unification techniques in general, has not previously been applied to the problem of discovering replacement functionality for breaking changes to APIs. While other work has found it useful for generalizing equations, code clone detection, or small-scale code reuse (see Section 6.1.1), it would be reasonable to expect that the problem of detecting replacement functionality in response to API evolution might pose unique challenges for anti-unification and related techniques. As a preliminary step to building tool-support based

```
FIND-BEST-FIT(cast)
 1  List ← ()
 2  foreach n ∈ cast
 3       do foreach pm ∈ pm[n]
 4            do APPEND(pm, List)
 5  SORT(List)
 6  while size[List] ≠ 0
 7       do head ← POP(List)
 8          isBest[head] ← TRUE
 9          src ← src[head]
10          dst ← dst[head]
11          foreach pm ∈ pm[src]
12               do if pm ≠ head
13                    then DELETE(pm,pm[src])
14                         DELETE(pm,List)
15          foreach pm ∈ pm[dst]
16               do if pm ≠ head
17                    then DELETE(pm,pm[dst])
18                         DELETE(pm,List)
19  return cast
```

on MSG, I conducted a study to evaluate its effectiveness in resolving known binary incompatible changes to APIs. Specifically, I was interested in answering three research questions.

RQ 6.1: Can MSG correctly resolve breaking changes between library API versions?

RQ 6.2: Do the assumptions and simplifications (see Section 6.2) impact its effectiveness?

RQ 6.3: How does MSG compare to alternative analysis techniques?

To answer these questions, I applied MSG to recommend replacement functionality for method-level binary incompatible changes in the APIs detailed in the Library Migration Corpus ("the Corpus", see Section 5.2). MSG was applied across a total of 13 transitions between API versions.

APACHE STRUTS[3] versions 1.0.2, 1.1.0, 1.2.4, 1.2.6;[4]

[3] http://struts.apache.org/

[4] The 1.2.7–1.2.8 and 1.2.8–1.2.9 migrations were omitted due to lack of binary incompatible changes.

LOG4J[5] versions 1.0.4, 1.1.3, 1.2.1, 1.2.2, 1.2.5, 1.2.6, 1.2.8, 1.2.9;

JDOM[6] versions 1.0.b6, 1.0.b7, 1.0.b8, 1.0.b9;

For each API migration analyzed, MSG was applied to make change recommendations for the recorded binary incompatible methods. For BI API change I examined MSG's recommendation of the appropriate method replacement in the next API version for that library. If the recommendation agreed with the Corpus, I marked it as correct; if the recommendation disagreed, but on further investigation it represented a still viable replacement method for the breaking change, I also marked it as correct; and, if the recommendation was wrong, I recorded it as incorrect.

In some cases, changes were made to an API such that no replacement functionality existed in the new API version; in these cases, developers are expected to either: (i) provide an implementation of functionality required by the API (e.g., addition of methods to an interface, change of a concrete method implementation to an abstract method implementation); or (ii) modify their usage of the API such that they no longer rely on that functionality. In these cases, MSG should reported that no replacement functionality exists: if this was true, I recorded it as correct; however, if a replacement did exist, I recorded the recommendation as incorrect.

6.3.1 Observations

Table 6.1 presents a summary of recommendations for the STRUTS, JDOM, and LOG4J libraries. The table records the count of method-level binary-incompatible changes for each API migration, along with the number of correct recommendations provided by each of six techniques: MSG and five of the potentially-automatable approaches detailed in the empirical study presented in Chapter 5. For convenience, I re-summarize these techniques here.

Call analysis. Examines how the library's source code migrates its own usage of the API between versions, and suggests adaptations from these changes.

[5] http://logging.apache.org/log4j/1.2/
[6] http://www.jdom.org/

Transition	*BI methods*	*MSG*	*Call analysis*	*Explicit in comments*	*Text search on comments*	*Text search on entity name*	*Release documents*
LOG4J							
v1.0.4–v1.1.3	16	6	8	1	0	2	0
v1.1.3–v1.2.1	96	31	28	42	1	12	53
v1.2.1–v1.2.2	1	1	0	0	0	1	0
v1.2.4–v1.2.5	1	0	0	0	0	0	0
v1.2.5–v1.2.6	1	0	0	0	0	0	1
v1.2.8–v1.2.9	1	1	1	0	0	0	0
overall		34%	32%	37%	1%	13%	47%
JDOM							
v1.0.b6–v1.0.b7	14	8	4	2	0	2	1
v1.0.b7–v1.0.b8	30	5	2	18	0	0	0
v1.0.b8–v1.0.b9	19	1	11	3	0	2	0
overall		22%	27%	37%	0%	6%	2%
STRUTS							
v1.0.2–v1.1.0	75	5	6	6	5	19	1
v1.1.0–v1.2.4	96	27	6	34	5	25	27
v1.2.4–v1.2.6	1	1	0	0	0	0	0
overall		19%	7%	23%	6%	26%	16%
microevaluation		24%	19%	30%	3%	18%	24%
macroevaluation		25%	22%	32%	2%	15%	21%

Table 6.1: Comparison of MSG against other inference techniques.

Explicit in comments. Records whether inline comments associated with the affected entities indicated their replacements.

Text search on comments. Records whether using an inline comment as a textual search query yields a correct replacement.

Text search on entity name. Uses textual search on the name of the entity to find related code in the new library.

Release documents. Captures whether a change and its replacement functionality was documented in the release notes, migration guides, or library documentation.

It is important to note that these five techniques were applied manually by me prior to undertaking this study, as I was evaluating the potential of each approach if it were implemented in an ideal fashion. As such, these values should be taken as upper bounds that may be difficult to achieve by an automated tool. I also do not include the *Implementation details* or the *Experience* technique described in Section 5.1.2, as these techniques involved extensive human reasoning, experience, or serendipity in their analysis, which is unlikely to be automatable.

In total, 351 method-level, 1:1 binary-incompatible changes were analyzed across the 12 API transitions in the study. I calculated correctness via two methods: (1) *microevaluation*, in which all binary-incompatible changes are given equal weight and the correctness is simply the percentage of overall correct recommendations; and (2) *macroevaluation*, in which the correctness is calculated for individual systems and the overall correctness is an average of these. Macroevaluation accounts for the differences in systems, and the differing numbers of changes for each.

6.3.2 RQ 6.1, 6.3: Can MSG resolve changes comparatively well?

The initial results in Table 6.1 are not promising at first glance: while MSG is able to identify replacement functionality for API change (addressing RQ 6.1), the accuracy of MSG appears to be roughly the same as best results from other techniques (which addresses RQ 6.3).

A case-by-case examination of the results though reveals that different techniques work in different cases. In 86 of the 351 cases, none of the techniques provided a correct answer; only one technique provided a correct answer in 141 cases; two techniques in 101 cases; three techniques in 20 cases; four techniques in 3 cases; and five or more techniques in 0 cases. There were 20 cases in which only MSG provided the correct recommendation. The large variation in results between systems also tells us that details of the systems are important: for example, it should be obvious that the call analysis technique cannot work in cases where the library does not make use of its own methods (i.e., it is not self-referential). This suggests that MSG may offer a unique contribution in its analysis.

6.3.3 RQ 6.2: Does MSG's assumptions impact its effectiveness?

Classification	Occurrences	Classification	Occurrences
Selection	309	Delegation	96
Structural ambiguity	203	Merge/split code	33
Conceptual reworking	168	External API	30
		Other	28

Table 6.2: Categories of change characteristics that MSG fails to resolve.

To answer this question, I examined all 265 cases where MSG failed to recommend the correct replacement, in order to characterize:

1. whether the method's original implementation presented difficulties for MSG's analysis;
2. how the functionality evolved across the API versions, and if that conflicted with MSG's assumptions; and,
3. whether the MSG algorithm was at fault in finding the correct replacement.

I then grouped the results into categories of reasons for the failures; these categories and their frequencies are shown in Table 6.2. As many changes had multiple reasons for failure, the sum of the occurrences is greater than the number of failure cases.

I found that the majority of the failures were caused by changes exhibiting one of six major characteristics:

1. **Selection.** The MSG algorithm is designed with the assumption that, if the method's enclosing class exists in the next API version, matching functionality is more likely to be found within that class. As this is not always the case, MSG can confuse methods that are structurally similar to the original method in that class, despite the existence of better matches elsewhere.

2. **Structural ambiguity.** Best practices and development habits have led to the creation of several method archetypes, such as "getter" and "setter" methods, that are structurally common, resulting in numerous spurious matches with the new API version. Furthermore, the limited amount of structural detail therein, combined with a high degree of lexical similarity, makes these difficult to distinguish.

3. **Conceptual reworking.** In several cases, the implementation of methods was radically altered in the next version of the API, typically to accommodate a significant conceptual change. As a result, the structure of the replacement functionality bears little resemblance to the method it replaces, falsely suggesting that the methods are not related.

4. **Delegation.** There were several instances where library maintainers delegated functionality in the old API to its intended replacement, most likely as a precursor to its eventual removal. As a consequence, the implementation of the affected method comprises a single method call, and thus is structurally unlike the implementation of the method being delegated to.

5. **Merge/split code.** Merges and splits of methods present a challenge for MSG, even where a 1:1 replacement is possible; the degree to which the original method was split up or merged with other functionality proportionally dilutes the structural correspondence.

6. **External API.** There were some cases in which functionality was removed from an API, with the expectation that developers were to use the equivalent method within an additional library. The initial evaluation only analyzed the affected library source code, and did not consider other external libraries.

A few cases did not fit in these categories; due to their relative infrequency, I do not address them in this work.

The results suggest that while the assumptions and simplifications made in Section 6.2 do not negatively the performance of MSG, some characteristics of library change prove confusing to this technique. However, this initial evaluation suggests several avenues for improvement to MSG: how replacement methods are selected, how delegation is resolved, and how structurally ambiguous method implementations are analyzed could be simple problems to address. Other failure categories pose a more significant challenge: it may be too difficult to attempt to resolve cases through structural comparison when the underlying structures are heavily altered between successive versions, and this may be a fundamental limitation to this approach.

That said, *Selection*, *Structural ambiguity*, and *Delegation* describe the majority of issues MSG encountered in the evaluation; addressing these three limitations could resolve roughly 70% of the problems encountered in recommending library adaptations.

6.4 UMAMI: API-Aware Structural Matching

UMAMI[7] is a prototype API change recommendation tool that looks for matches between the structural generalizations of methods in two versions of a library, in order to suggest replacement functionality for methods removed from the API between versions. UMAMI is implemented as a plugin to the ECLIPSE IDE, and makes use of the ECLIPSE JAVA DEVELOPMENT TOOLKIT[8] to extract AST representations of JAVA source code. The tool takes as input the source code of two versions of a library, set up as projects within the ECLIPSE IDE, and attempts to identify equivalent functionality in the new API version for methods present in the original library. For each method in the older library version, UMAMI performs a series of pair-wise comparisons to all methods in the newer library version; since UMAMI is not aware of what is or what is not part of the API (aside from visibility restrictions), nor what has changed, it attempts to find matches for all methods in the original library to their equivalents in the new version.

[7] *Umami* is a loanword from Japanese designating one of the five basic tastes, which typically derives from the presence of the food additive monosodium glutamate (MSG).

[8] http://www.eclipse.org/jdt

For a given method, UMAMI begins by constructing two sets of possible replacement methods: the first set includes methods whose enclosing classes share both a similar name with its enclosing class, and have a number of methods common to both classes, and thus may be more likely to contain the replacement; the second set of candidate replacements is created by pairwise generalization of all methods that were not included in the first set. For each generalized candidate replacement (regardless of which set it belongs too), the candidates are ordered by their similarity value, and then a greedy selection is applied based on their similarity such that each method has at most one generalization candidate. Once all methods in the original library have been matched to their best generalization candidate in the new library, UMAMI generates the set of proposed mappings, along with its estimated similarity value to provide a measure of the strength of its recommendation. These mappings can then be loaded into another tool, or perused by a developer who is attempting to resolve a specific API change.

6.4.1 Engineering MSG for APIs

During pilot studies, three classes of API change proved problematic for the MSG approach, but two of these are simple to account for.

6.4.1.1 Delegation

In some cases, a library will delegate functionality from a method soon to be removed to its eventual replacement. But, applying a structural analysis of that delegated method erroneously looks for matches to the delegating call itself, rather than matches to the delegated method, causing structural comparisons where one method has only a single line of code. To address this, an additional check was added to see if a method may contain only a delegated call and, if so, use the delegated call instead.

6.4.1.2 Getter/Setter

Most getter/setter methods are simple wrappers for variable assignment and return statements. The archetypal nature of these methods results in widespread structural commonalities that makes

individual getter/setter methods structurally indistinguishable from each other. However, the fields accessed by such methods, and their declaring class, provide some differentiating context. To leverage this, the field declaration and the class declaration are included as additional points of structural comparison when matching such methods across versions. Before the getter or setter method is compared against a method from the second version, the possible replacement method structure is checked to see if it is a getter or setter to determine if the extra contextual information should be included in the comparison.

6.4.1.3 Addition to an Interface

When an interface type has an operation added to it, all classes implementing that interface must add a method implementation for that operation. There is no reason to expect that an existing method implementation will correspond to this required method addition, and so, it is simply marked as a needed addition.

6.4.2 The UMAMI algorithm

The UMAMI algorithm shows the resulting modifications to the original approach presented in Chapter 6; the UMAMI-HELPER procedure is a helper procedure used to simplify the presentation. UMAMI first constructs a set of fully qualified names (M_A and M_B) for every method declaration in each version, respectively (lines 3–4). The complement of the set from the first version with the set from the second version is the set of methods that have been removed between versions (line 5). The complement of the second version with the first version are new additions to the library (line 6). For each (removed) method in A_{removed} (line 7), two special cases are tested for: delegation (line 8) and getter/setter (UMAMI-HELPER, lines 3, 6–7).

For getters/setters, FIND-MATCHES is called with arguments that broaden the possibilities: the field operated on is considered, as is the signature of the class containing the getter/setter. For methods that are neither a delegator nor getter/setter, the set of candidate replacements are constructed by generalizing the methods with the complete set of methods from the second version

```
UMAMI(A, B)
 1  R ← ∅
 2  D ← ∅
 3  M_A ← methods[A]
 4  M_B ← methods[B]
 5  A_removed ← M_A \ M_B
 6  B_added ← M_B \ M_A
 7  foreach m_A ∈ A_removed
 8       do if IS-DELEGATOR(m_A)
 9            then D ← D ∪ (m_A, delegate[m_A])
10            else  R ← R ∪ UMAMI-HELPER(m_A, M_B)
11  foreach (m_A, delegate) ∈ D
12       do d ← {delegate} ∩ M_A ∩ M_B
13          if d ≠ ∅
14            then R ← R ∪ (m_A, d)
15            else  d ← {delegate} ∩ R
16                  if d ≠ ∅ then R ← R ∪ (m_A, d)
17  foreach m_B ∈ B_added
18       do if IS-PARENT-INTERFACE(m_B) ∧ m_B ∉ R
19            then R ← R ∪ (NIL, m_B)
20  return R
```

```
UMAMI-HELPER(m_A, M_B)
1  cast ← ∅
2  foreach m_B ∈ M_B
3       do if ¬IS-GET-SET(m_A) ∧ ¬IS-GET-SET(m_B)
4            then c ← FIND-MATCHES(m_A, m_B)
5                 cast ← cast ∪ c
6            else  c ← FIND-MATCHES({sig[class_mA], field_mA, m_A},
                                    {sig[class_mB], field_mB, m_B})
7                 cast ← cast ∪ c
8  return (m_A, FIND-BEST-FIT(cast))
```

(UMAMI-HELPER, lines 4–5). Candidate replacements that do not have a similarity of at least 25% with the removed method are dropped from consideration (not shown). The set of candidate replacements are ordered by similarity then by locality (i.e., is the replacement in the same class, package, or project).

An additional minor adjustment catches breaking changes due to the addition of method declarations to interface types, which require that method implementations be added to any of their subtypes. Once the delegation and getter/setter cases are resolved, the set of new functionality is analyzed specifically for this case (lines 17–19).

6.5 Evaluation of Umami

I am interested in answering three questions.

RQ 6.4

Can UMAMI correctly resolve breaking changes between library API versions?

RQ 6.5

How does UMAMI compare to alternative analysis techniques?

RQ 6.6

How does UMAMI compare to existing API change recommender tools?

To answer these questions, I applied UMAMI to recommend replacement functionality for method-level binary incompatible changes in the APIs detailed in the Library Migration Corpus from Section 5.2 ("the Corpus"). UMAMI was applied across a total of 13 transitions between API versions.

APACHE STRUTS[9] versions 1.0.2, 1.1.0, 1.2.4, 1.2.6;[10]

LOG4J[11] versions 1.0.4, 1.1.3, 1.2.1, 1.2.2, 1.2.5, 1.2.6, 1.2.7, 1.2.8, 1.2.9;

JDOM[12] versions 1.0.b6, 1.0.b7, 1.0.b8, 1.0.b9;

For each binary incompatible method, I examined UMAMI's recommendation of the appropriate method replacement in the next API version for that library. If the recommendation agreed with

[9] http://struts.apache.org/
[10] The 1.2.7–1.2.8 and 1.2.8–1.2.9 migrations were omitted due to lack of binary incompatible changes.
[11] http://logging.apache.org/log4j/1.2/
[12] http://www.jdom.org/

the Corpus, I marked it as correct; if the recommendation disagreed, but on further investigation it represented a still viable replacement method for the breaking change, I also marked it as correct; if the recommendation was wrong, I recorded it as incorrect; and, if UMAMI gave a recommendation that no replacement functionality existed, when in fact it did, I marked it as incorrect.

In some cases, changes were made to an API such that no replacement functionality existed in the new API version; in these cases, developers are expected to either:

1. provide an implementation of functionality required by the API (e.g., addition of methods to an interface, change of a concrete method implementation to an abstract method implementation); or,

2. modify their usage of the API such that they no longer rely on that functionality.

UMAMI attempts to detect such changes to the API, and in those cases its recommendation is that no replacement functionality exists: if this was true, I recorded this as correct; however, when a replacement did exist, I recorded this recommendation as incorrect.

Because of my experience with the libraries comprising the Corpus, I felt that there was a risk the approach may be inadvertently over fitted to the particular idiosyncrasies of the STRUTS, LOG4J, and JDOM libraries and their change history. To address this issue, I prepared an additional set of library migration data, using the same techniques outlined in Section 5.1 to further evaluate UMAMI.

I chose to analyze APACHE LUCENE,[13] a popular text-search engine library, and determined the correct migrations for binary incompatible changes stemming from the following transitions:

APACHE LUCENE versions 3.1.0, v3.2.0, v3.3.0, v3.4.0.

I identified a total of 42 method-level binary incompatible changes across the three API migrations, and determined:

[13] http://lucene.apache.org/core

1. the correct replacement for each change; and,

2. whether each of the five change recommendation techniques detailed in Section 5.1.2 were also effective in resolving the change.

After the dataset was constructed, and the results of the other analysis techniques known, I then applied UMAMI to the LUCENE data, and compared its recommendations to the correct adaptations.

6.5.1 Performance

UMAMI was run on a machine using an INTEL i7-2630QM processor (4 cores, 2 GHz clock speed) with 8 GB RAM, a solid-state disk drive, and running MICROSOFT WINDOWS 8: for the two most difficult library migrations in terms of both size and number of binary incompatible changes introduced (STRUTS v1.0.2–v1.1.0, and v1.1.0–v1.2.4), UMAMI was still able to complete its analysis in 31 seconds. For generating an individual recommendation, the average case required 214 ms, and the worst case required 4.1 s.

6.5.2 RQ 6.4, 6.5: Can UMAMI resolve changes comparatively well?

Table 6.3 presents a summary of recommendations for the LOG4J, JDOM, STRUTS, and LUCENE libraries. The table records the count of method-level binary incompatible changes for each API migration, along with the number of correct recommendations provided by UMAMI and the five other API change analysis techniques described in Section 5.1.2, for which I provide a brief refresher.

Call analysis. Examines how the library's source code migrates its own usage of the API between versions, and suggests adaptations from these changes.

Explicit in comments. Records when inline comments associated with the method indicated a replacement.

Text search on comments. Records when a lexical search on the method's inline comment yields a replacement.

Text search on entity name. Uses lexical search on the method's signature to find a replacement method.

Release documents. Records if a change and its replacement functionality were documented in release notes, migration guides, or library documentation.

For each of the textual search techniques, I constructed an automated lexical analyzer to evaluate the effectiveness of edit-distance-based (specifically, the Levenshtein distance) lexical analysis on method signatures and inline comments, as used by other change recommendation approaches [Wu et al., 2010; Schäfer et al., 2008]. For inline comments, I used the QDOX library[14] to extract the comments associated with every JAVA element, and performed a series of pairwise comparisons between every binary-incompatible method from the old API, and all other methods in the new API. For each pair of method comments, the number of single-character edits needed to transform the original comment into the comparison comment was calculated, and the lowest edit distance was used to chose the best match. A similar approach was used to compare entity names, except that the fully qualified names of the methods, including the typed parameter list, was used instead of comments.

It is important to note that the *Call analysis*, *Explicit in comments*, and *Release documents* techniques were applied manually by a human being in order to evaluate the potential of each approach if it were implemented in an ideal fashion. As such, these values should be taken as upper bounds that may be difficult to achieve by an automated tool. I do not include the *Implementation details* technique I applied earlier in Section 5.1.2—in which a series of static and data-flow analyses were applied in conjunction with human reasoning to determine the intent behind the implementation and where equivalent code may exist in the new library—as it seems unlikely to be automatable;

[14] http://qdox.codehaus.org/

likewise I do not include the *Experience* technique (see Section 5.1.2), which involved the experimenter stumbling across the answer through serendipity.

In total, 393 method-level binary-incompatible changes were analyzed across the 16 API transitions in this study. I calculate correctness via two methods.

1. *Microevaluation*, in which all binary-incompatible changes are given equal weight and the correctness is simply the percentage of overall correct recommendations.

2. *Macroevaluation*, in which the correctness is calculated for individual systems and the overall correctness is an average of these. Macroevaluation accounts for the differences in systems and the differing numbers of changes for each.

UMAMI's application of matching via structural generalization is able to resolve a significantly higher number of changes than the other techniques surveyed: 56% of all breaking-changes between API versions were resolved using UMAMI, as compared to 22% to 27% of changes using other analysis techniques, addressing questions RQ6.4 and RQ6.5. I further note UMAMI's effectiveness is even higher when resolving changes for the new LUCENE dataset than it was for the original systems studied: this implies that UMAMI is not heavily tied to the specific change characteristics of LOG4J, JDOM, or STRUTS.

The results also suggest that there is considerable orthogonality in the changes resolvable using UMAMI over the other techniques examined: for 17% of the changes examined, UMAMI was the only technique able to correctly identify a replacement. While there were other changes resolvable using only one of the other techniques, the numbers were significantly smaller: 6% were only resolvable using release documentation, 6% only though the documented replacement in the source comments, and 3% only through call analysis.

6.5.3 RQ 6.6: How does UMAMI compare to other API change recommenders?

Many, if not most, of the research prototypes in this area use a combination of techniques to overcome limitations or disadvantages of each. For example, Wu et al. [2010] supplement call

analysis with lexical analysis to identify possible replacements when no usage examples exist, while Dagenais and Robillard [2011] and Schäfer et al. [2008] require additional sources of data to glean usage examples by either: (a) incorporating incremental change sets from the library's revision history into the analysis pool; or, (b) adding source code from other successfully migrated programs. Such approaches may offer additional improvements that outperform UMAMI.

To understand the effectiveness of UMAMI relative to other API change recommender approaches, I compared the results of three other change recommenders that were previously used find replacements for API changes in the Struts 1.1.0 to 1.2.4 migration, which is the only one for which data exists for all of them: AURA [Wu et al., 2010], Schäfer et al. [Jonas, 2007; Schäfer et al., 2008], and REFACTORINGCRAWLER [Dig et al., 2006].[15] While two previous studies have compared the relative effectiveness of some of these techniques [Wu et al., 2010; Schäfer et al., 2008], the studies were undertaken without *a priori* knowledge of the actual changes in that migration, and thus can only state where each tool agrees or disagrees with each other. To improve on this, I acquired the recommendation data from each of the authors on the actual predictions made for API changes and replacements for the Struts migration. I then compared these predictions to the list of binary incompatible changes recorded in the Corpus, and determined if their proffered replacement matched the known change. In cases where there was disagreement, I re-investigated the Struts source code to determine if the disagreement represented an equally valid substitution or an error in the Corpus, rather than an error in the recommender.

Table 6.4 compares UMAMI's effectiveness to three other API change recommendation techniques: for each technique, I recorded the number of binary incompatible changes for which the technique correctly resolved (True Positive, or TP), issued a wrong recommendation (False Positive, or FP), or did not make a recommendation (False Negative, or FN). The respective precision, recall, and F-score of each approach was then calculated. The results reported here are different than those of prior works as:

[15] A comparison with the HIMA tool of Meng et al. was not done as I was unaware of that work at the time of this research. A comparison of the HIMA tool's effectiveness to these techniques can be found in Appendix D.

1. only recommendations for binary incompatible method changes to the API are considered; and,

2. the set of all binary incompatible changes is known for this migration.

The results show that UMAMI offers a general improvement over the results of the AURA tool: UMAMI is able to correctly recommend changes in 42% more cases, and was capable of making recommendations for all BI changes. The comparisons to Schäfer et al. and REFACTORING-CRAWLER are more varied: both of these techniques achieved high precision at the expense of recall, and thus are useful in less than a third of the cases that UMAMI is. However, by comparing the F-score's weighted averaging of each technique's precision and recall it becomes apparent that UMAMI offers a significant overall improvement in accuracy, and thus effectiveness. Interestingly, while both AURA and Schäfer et al.'s approach are based on call analysis, their implementations vary in their effectiveness. Schäfer et al. discovers four methods using call analysis on the Struts library implementation that AURA does not; AURA discovers two methods via call analysis that Schäfer et al. does not, but makes the remainder of its discoveries using lexical analysis.

Overall, UMAMI proves to be more effective in recommending correct replacements for binary incompatible API changes than any of the compared techniques. Because UMAMI represents a fundamentally different technique in recommending API migrations than these approaches (i.e., matching via structural generalization, rather than call analysis, or shingle similarity combined with heuristics), I interpret this as showing that MSG is not only useful as a change inference technique, it could be used in concert with these and other techniques to further improve its effectiveness.

6.6 Discussion

6.6.1 Threats to Validity

There are several threats to the internal validity of this work. First, the extensions made to MSG to form UMAMI may have overfit to the libraries under analysis. To avoid this possibility, I created an additional verification dataset based on the LUCENE API to see if UMAMI was only effective when examining the original three libraries; in fact, the results in Table 6.3 show that UMAMI was *most* effective on the new API.

Secondly, as I determined the correct replacements for the binary-incompatible API changes, I may not have been objective in analyzing the results. For the original Corpus APIs (STRUTS, JDOM, LOG4J), the replacements for binary incompatible changes were determined prior to the idea of performing this current study, and the resulting Corpus was publicly available. For the extension to LUCENE, the replacements were determined prior to collecting the results from UMAMI, but I acknowledge that my experience could have introduced bias. I will be adding the LUCENE data to the Corpus so others may inspect and use these results.

There are three potential issues regarding the construct validity of the experiments. The first was that UMAMI was deemed correct if the expected recommendation was amongst the top 3 results that it returned. The choice here was consistent with those of other researchers in API migration recommendation (e.g., [Dagenais and Robillard, 2011]). The second potential issue is that the correct recommendations of UMAMI may have been merely fortunate accidents. To check this, as part of the evaluation I investigated correct recommendations to verify that the result made sense in the context of what the algorithms should produce. I also investigated UMAMI's mistakes, and found that when it failed to correctly resolve the change, it was clear that it failed either because of an assumption made in the construction of the tool, or due to a fundamental limitation of structural generalization. In other words, UMAMI failed to work when it *should* fail to work.

The work suffers from three issues with respect to external validity. First, the evaluations have examined method-level changes on 16 versions of four JAVA-based APIs; however, I cannot say

if these are representative of all JAVA-based APIs, nor can I say that the migrations analyzed are typical of how JAVA libraries evolve. Thus, I cannot say whether UMAMI will be equally capable when examining other libraries. However, these results are still promising: the evaluations show that MSG can be useful in matching broken functionality to its replacement in a new API version, and that it is capable of providing correct recommendations when other approaches cannot.

Finally, the evaluation of UMAMI was limited to API changes for which a one-line code replacement existed, and did not consider changes where 1:n, n:1, or m:n lines of code were altered (see Section 5.1.3), as the tool is not yet capable of this. While I believe this is not a fundamental limitation of this approach (see Section 6.2.2), it is a limitation of this evaluation: I cannot say whether MSG or UMAMI would work equally as well in identifying API changes where the replacement was not a single line of code for a single line of code. However, the comparison of UMAMI to AURA did not suffer from this restriction: in their work, Wu et al. [2010] indicate that AURA is able to capture library changes were 1:n and n:1 transformations took place. However, AURA detects *no* such changes to the STRUTS 1.1.0 to 1.2.4 library, and thus misses *all* of the 1:n changes that were identified in the Library Migration Corpus.

6.6.2 Comparison with Cottrell et al. [2008]

UMAMI applies an approach developed by Cottrell et al. [2008] for supporting small-scale, pragmatic reuse [Holmes and Walker, 2007] of source code, and formed part of the implementation of their JIGSAW tool. Small-scale reuse involves identifying a portion of source code that implements functionality which is of interest to a developer, and understanding how this can be directly incorporated into the developer's software: often, developers would simply "cut-and-paste" the desired source code into a portion of a method they wished to add it to, and then spent time reworking the pasted example to fit the type declarations, inheritance hierarchies, and control-flow structures that were already in place. Cottrell et al. [2008] developed a "smart" copy-and-paste, where the context in which code was pasted into was identified and compared to the copied code to determine if there were equivalencies that could be semi-automatically remapped to reflect what was already present

in the new context. Their application of matching via structural generalization was developed to this end: identifying code which was similar allowed JIGSAW to suggest to the developer possible remappings that could ease its integration into the software.

The application of MSG in the UMAMI tool is novel to the problem of API change recommendation. The original context the technique was developed for involved a pre-defined snippet of code, which was to be compared against a target location which was missing the functionality the snippet should supply; in essence, MSG was applied to determine how the code snippet "fit the hole" present in the developer's source code. My application here uses MSG to identify structurally equivalent code between one snippet representing old API functionality, across thousands of possible examples present in the new library version. The problems in each case are quite different, but this work shows how MSG can be successfully adapted to this problem context.

6.6.3 Future Work

Several avenues for future work have become apparent during the investigations. Most obviously, MSG was useful in detecting changes that many existing approaches cannot; hybridizing UMAMI with additional analysis techniques may provide further improvements in effectiveness. I also observed some cases in Section 5.2 where the migration path for clients of an evolved API would not be simple one-for-one replacements, but general m:n cases. Structural comparison could be extended to these more general cases by, for example, finding multiple candidate matches to portions of methods. In addition, although UMAMI has been implemented to operate over ASTs, the approach could be generalized to other structured information, including semantic information. Whether general graphs could be matched as easily as trees, whether the cost of constructing the information would be prohibitive, whether the results would be an improvement—all remain to be seen.

6.7 Summary: Using Structural Analysis for Reactive Change

I have presented a new inference technique, *matching via structural generalization* (MSG), for recommending migration paths between API versions. MSG utilizes the syntactic similarity of the versions' implementations, augmented with simple knowledge of the semantic equivalence of differing syntactic structures, to detect correspondences between functionality in each library version.

This approach has been reified in a novel prototype API change recommender, called UMAMI, which given two versions of a library systematically compares the structural similarity of all methods in one version to all methods in the second, and greedily matches to the method with the highest structural similarity. In doing so, UMAMI identifies replacements for missing or altered API functionality, based on its structural similarity to a candidate replacement in the new library API. I evaluated UMAMI by comparing its results against the expected results from a benchmark data set, which I expanded to transitions within another library.

I found that UMAMI provides a correct recommendation in 56% of cases, which is approximately double that of the existing analysis techniques—including *call analysis*, *lexical search*, and *documentation analysis*. Furthermore, UMAMI offers improved precision and/or recall in comparison to other API change recommendation approaches. MSG is uniquely suited to some kinds of change analysis, as it was the only technique to recommend a correct replacement in 17% of cases examined, making it complementary to existing analysis techniques: future hybridized API change recommenders should leverage this new approach in combination with several of the existing approaches.

Table 6.3: Comparative results. U = UMAMI; BI = binary incompatibilities on methods; CA = call analysis; EC = explicit in comments; TSC = text search on comments; TSN = text search on entity name; RD = release documents.

Transition	BI	U	CA	EC	TSC	TSN	RD
LOG4J							
v1.0.4–v1.1.3	16	12	8	1	5	7	0
v1.1.3–v1.2.1	96	49	28	42	13	24	53
v1.2.1–v1.2.2	1	0	0	0	0	0	0
v1.2.4–v1.2.5	1	0	0	0	0	0	0
v1.2.5–v1.2.6	1	0	0	0	0	0	1
v1.2.8–v1.2.9	1	1	1	0	0	0	0
overall		53%	32%	37%	16%	27%	47%
JDOM							
v1.0.b6–v1.0.b7	14	8	4	2	8	10	1
v1.0.b7–v1.0.b8	30	12	2	18	5	8	0
v1.0.b8–v1.0.b9	19	14	11	3	2	1	0
overall		54%	27%	37%	24%	30%	2%
STRUTS							
v1.0.2–v1.1.0	75	34	6	6	10	9	1
v1.1.0–v1.2.4	96	51	6	34	29	42	27
v1.2.4–v1.2.6	1	0	0	0	0	0	0
overall		49%	7%	23%	23%	30%	16%
LUCENE							
v3.1.0–v3.2.0	3	3	3	0	0	0	1
v3.2.0–v3.3.0	27	27	12	0	4	7	23
v3.3.0–v3.4.0	12	10	5	0	2	1	0
overall		95%	48%	0%	14%	19%	57%
microevaluation		56%	22%	27%	20%	28%	27%
macroevaluation		63%	28%	24%	19%	27%	30%

Table 6.4: Comparison to AURA, Schäfer et al., and REFACTORINGCRAWLER (RC) for STRUTS 1.1.0 to 1.2.4.

	BI methods	UMAMI			AURA			Schäfer et al.			RC		
		TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN
STRUTS													
v1.1.0–v1.2.4	96	51	45	0	36	41	19	9	0	87	19	0	77
Precision		53%			47%			100%			100%		
Recall		53%			38%			9%			20%		
F-score		53%			42%			17%			33%		

Chapter 7

Hybridizing Recommenders for Reactive Change

Despite the new inference technique presented in Chapter 6, there still is no single analysis technique that can be relied upon to make correct recommendations for the vast majority of changes. However, the empirical study in Chapter 5, and the evaluation of the UMAMI prototype in Section 6.5 indicate there is some degree of orthogonality in the changes a particular change recommender technique is capable of resolving. Intelligently combining individual techniques into a hybrid recommender may thus be effective, if we can determine when a technique's recommendation is the most pertinent for a given change.

In this chapter, I present an approach based on *flexible hybridization* of inference techniques for change recommendation. Unlike prior techniques that have combined inference techniques through a hard-coded, predefined combinations [Wu et al., 2010], I aim to support the ability of end-users to add (and remove) of inference techniques as they seem appropriate for their migration needs. Further, rather than predefine how these techniques interact, I wish to see them configured by their users, through the provision of either example migrations, or other forms of training data that to improve the effectiveness of the resulting change impact and redress tool support.

The rest of this chapter is organized as follows. I begin in Section 7.1 by discussing my mechanical implementation of several different API change recommendation inference techniques, which I use to provide a baseline effectiveness of different points of change. I continue in Section 7.2 where I introduce the idea of flexible hybridization of techniques, by combining individually poor inference techniques together through a machine learning approach, which can be trained through user-provided examples to recognize when a particular technique is effective. I evaluate the promise of this approach in Section 7.3, where I compare the effectiveness of three different machine learning techniques in combining these inference techniques for API change recommen-

dation. The validity of the case study's results are discussed in Section 7.4, and I summarize the chapter in Section 7.5.

7.1 Evaluating Inference Techniques

Every change recommender for API migration takes as input two versions of a software library's API and/or its underlying implementation. The goal in constructing the recommender is that it analyze this input to suggest what the replacement functionality should be for entities removed from or altered in the older API version. I assume that, in most cases, functionality that disappears from a particular API is actually still present in some form in the new library version [Godfrey and Zou, 2005]: e.g., the functionality may be moved or renamed, replaced by alternative yet equivalent entities, or incorporated into existing API functionality (e.g., see Section 5.1.3). The issue then is finding an inference technique that is able to identify where the replacement functionality is, and thus the correct adaptation(s) necessary to migrate between API versions. But, if the individual inference techniques available are not able to resolve all changes to a specific API (see Section 5.1.1), can they be combined into a hybridized change recommender where the strengths of one technique can overcome the weaknesses of another, and thus be more effective as an ensemble? To study this, I explore two research questions:

RQ 7.1 How effective are individual inference techniques in resolving API change; and,

RQ 7.2 Could an end-user configure their combination to take advantage of their results?

I examine these questions through two empirical studies using data derived from the Library Migration Corpus (see Section 5.1.1: "the Corpus"). Before exploring possible hybrid approaches, I begin by describing in this section how I implemented five inference techniques, and evaluated their accuracy in inferring adaptive changes, with the goal of having a baseline with which hybrid approaches can be compared against. Later in Section 7.2, I present three approaches for hybridizing these inference techniques into an API change recommender.

7.1.1 Baseline Implementations

Evaluating the effectiveness of inference techniques is useful to understand if subsequent hybridizations offer real improvements, and to appreciate the particular capabilities of each. Previous empirical research on library and API evolution has *manually* examined the effectiveness of inference techniques in resolving breaking changes [Gosling et al., 2012, Ch. 13] to JAVA APIs [Cossette and Walker, 2012], and has concluded that many techniques are effective for only a small portion of the changes they analyzed. However, this work was limited in two ways: (1) the analysis only indicated whether the inference technique could resolve the incompatibility, and did not discuss the precision of the examined techniques; and (2) the manual processes meant some techniques used human reasoning to resolve cases that would otherwise be difficult to programmatically analyze, representing an upper-bound on the technique.

To improve on these limitations, I conducted a new empirical evaluation of how effective various inference techniques can be in resolving changes to APIs in software libraries, by creating a testbed that implements these techniques and applies them to a series of library migrations.

7.1.1.1 Methodology

I studied how well inference techniques can recommend replacement functionality for method-level *binary incompatible* changes to the APIs of JAVA libraries. A binary incompatibility (BI) between two APIs is defined as any change to a type such that code compiled and linked without error to a binary containing that type prior to the change, does not link to a new binary containing the changed type [Gosling et al., 2012, Ch. 13]; lists of BI categories are available online.[1] To simplify the implementation of the above inference techniques, I restricted the analysis to only method-level BIs in the API; however, to be clear as to their limitations, they were applied to all method level changes, although their implementation targeted 1:1 changes.[2]

For the study, I examined the effectiveness of these techniques in resolving BIs in the JAVA

[1] e.g., http://wiki.eclipse.org/Evolving_Java-based_APIs_2

[2] In the evaluation of structural similarity in Section 6.5, methods whose enclosing classes were BIs, and methods added/removed from interfaces were also considered; however, only changes of a 1:1 nature were analyzed.

APIs detailed in the Corpus from Section 5.1, specifically:

APACHE STRUTS[3] versions 1.0.2, 1.1.0, 1.2.4, 1.2.6;

LOG4J[4] versions 1.0.4, 1.1.3, 1.2.1, 1.2.2, 1.2.5, 1.2.6, 1.2.8, 1.2.9;

JDOM[5] versions 1.0.b6, 1.0.b7, 1.0.b8, 1.0.b9;

APACHE LUCENE[6] versions 3.1.0, v3.2.0, v3.3.0, v3.4.0.

In total, 16 transitions between API versions were examined. For each method-level binary incompatible change in the API, each inference technique was applied to the relevant source code or documentation, and all recommendations with a minimum similarity of 0.2 were recorded; this restriction was needed to manage the size of the data generated for larger libraries that underwent a significant number of changes, and for which a significant number of methods in the API existed. Recommendations for replacement functionality by each inference techniques were compared to the known replacements as previously documented in the Corpus, and I recorded the number of correct and incorrect predictions made by each, as well as the number of cases for which no prediction was made.

7.1.1.2 Implemented Techniques

I implemented five inference techniques from the literature on API migration, or on origin analysis systems, presented in Section 3.3.

- *Lexical edit distance (LED)* compared two strings and determines the number of character-based edits necessary to transform one string into another [Levenshtein, 1966; Schäfer et al., 2008; Wu et al., 2010]. It was normalized to a value between 0 and 1 by using the length of the original string, and then inverting the result such

[3] http://struts.apache.org/
[4] http://logging.apache.org/log4j/1.2/
[5] http://www.jdom.org/
[6] http://lucene.apache.org/core

that a value of 1 indicated the strings were identical. If the edit distance was greater than the length of the original string, I capped its similarity at 0.

- *Longest common substring (LCS)* compared two strings to determine the length of the largest continuous sequence of characters common to both strings (not to be confused with longest common subsequence, which does not require that the sequence be continuous [Kim et al., 2007]). The length of the LCS was also normalized to a value between 0 and 1 by using the length of the original string, and then inverting the result such that a value of 1 indicated the longest common substring was the entirety of the string, i.e, they were identical.

- *Deprecation and related comments* involved attempting to automatically recover references to replacement functionality from inline JAVA code comments associated with the original method. I used the QDOX library[7] to extract comments associated with the method, checking if a deprecation tag was present; if it was, I looked for a link tag indicating that replacement functionality was associated with this method prior to its removal, and recorded the associated method as the inferred replacement with a similarity of 1. If no link tag was present, I applied regular expressions to both the text associated with the deprecation tag, and the entirety of the code comment, to look for entities that resembled a reference to a JAVA method. Any potential matches were compared to functionality in the next API version to see if the reference was a spurious match or not; in some cases, only a method name was provided, and so all methods in the API that shared the same name were recorded as an inferred replacement. The similarity of each replacement was then calculated as being the inverse of the total number of such matches, or 1 if only one match was found.

- *Structural similarity* applied the UMAMI prototype from Section 6.4 to compare

[7] http://qdox.codehaus.org/

method declarations in the old API version to other methods in the new API by identifying common substructures between them via their ASTs. The degree to which each AST contains equivalent substructures (e.g., statements, logical tests, for-loops, etc.) is used to derive the overall similarity between the two methods. Additional details on the algorithms and procedures used to compare and evaluate ASTs were provided earlier in Section 6.2.2. By comparison, other approaches applying structural comparisons do exist, but have focused on the declaration and inheritance hierarchies, rather than implementation structural similarity [Xing and Stroulia, 2007b; Dagenais and Robillard, 2011].

- *Call analysis* compares the usage of API methods in the library's original implementation against the library's new implementation to discover migration examples for changes to the API [Schäfer et al., 2008; Wu et al., 2010; Dagenais and Robillard, 2011]. I implemented call analysis using the JAVA DEVELOPMENT TOOLKIT (JDT) provided with the ECLIPSE integrated development environment (IDE)[8] to extract references to methods in each version of the library's implementation, tracking the reference's enclosing caller, its calling order within the method, and any associated call chains or assignment operations. References to methods associated with binary incompatible methods in the next API version were then examined in both implementations, and updates were recorded as possible replacement functionality. The similarity of each replacement was then calculated as being the inverse of the total number of replacements identified, or 1 if only one match was found.

The suite containing the implementations of all the above inference techniques was run on a machine using an Intel i5-3570K processor (4 cores, 3.4 GHz clock speed) with 8 GB RAM; the time spent analyzing all 15 library migrations for all inference techniques was approximately 6 minutes.

[8] http://www.eclipse.org/

7.1.2 Observations

Table 7.1 presents a summary of recommendations for the STRUTS, JDOM, LOG4J, and LUCENE libraries. The table records the count of method-level binary-incompatible changes for each API migration, along with the number of correct recommendations provided by each of the seven techniques listed in Section 7.1. A total of 440 method-level binary incompatible changes were examined across 16 Java API migrations. For each binary incompatibility, only the best prediction (i.e., highest similarity) provided by the technique for that change is compared to the known answer, and evaluated as being correct or incorrect. In cases where incorrect recommendations are made, a second check is made to see if they may in fact refer to an alternative replacement that could also be valid, and thus should still be considered as correct; I did not find any additional valid replacements.

7.1.3 RQ 7.1: How effective are inference techniques in resolving API change?

The results show that each technique is highly variable in its effectiveness across APIs, and particularly across differing migrations; this is consistent with my previous, manually-derived results (see Section 5.3). Certain API migrations appear to be more difficult to analyze using these inference techniques (e.g., LOG4J 1.1.3 to 1.2.1, STRUTS 1.1.0 to 1.2.4), reflecting both the changes they underwent, as well as the degree of similarity in naming (and sometimes functionality) between multiple entities in the API [Cossette and Walker, 2012]. No single inference technique is able to adequately recommend correct change replacements for all BIs, and certainly not for every migration. In fact, across all APIs, the average recall rate for any particular inference technique was quite low; sometimes as little as 10%.

There is a relative degree of similarity in overall effectiveness between all four lexical inference techniques. For nearly all API migrations, the edit distance between two methods was a better predictor of a correct match than determining the LCS between them, although the benefits were often slight. In contrast, the LCS between two comments was often more useful in discovering

Table 7.1: Prediction results from the implemented inference techniques. Σ BIs = total binary incompatibilities, CA = call analysis, SS = structural similarity, LED = lexical edit distance, LCS = longest common substring, Sig = signature, Com = comments, DC = deprecation comments, T = true recommendations, F = false recommendations.

				CA		SS		LED: Sig		LED: Com		LCS: Sig		LCS: Com		DC	
API	Old	New	Σ BIs	T	F	T	F	T	F	T	F	T	F	T	F	T	F
JDOM	b6	b7	13	4	0	7	6	10	3	8	3	9	4	9	3	0	0
JDOM	b7	b8	29	3	0	2	27	7	22	5	6	6	23	9	20	0	0
JDOM	b8	b9	20	11	0	8	12	1	19	2	16	1	17	2	16	0	0
LOG4J	1.0.4	1.1.3	14	8	0	5	9	7	7	5	4	5	9	4	5	0	0
LOG4J	1.1.3	1.2.1	95	26	2	17	78	24	71	13	26	22	69	19	54	0	0
LOG4J	1.2.1	1.2.2	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
LOG4J	1.2.4	1.2.5	3	0	0	0	3	0	3	0	0	0	1	0	0	0	0
LUCENE	3.1.0	3.2.0	3	0	2	0	3	0	3	0	0	0	1	0	0	0	0
LUCENE	3.2.0	3.3.0	27	11	0	5	22	7	20	4	1	7	1	5	1	0	0
LUCENE	3.3.0	3.4.0	12	2	1	1	11	2	10	2	1	0	6	0	3	0	0
STRUTS	1.0.2	1.1.0	81	7	0	7	94	9	72	10	53	13	66	11	68	0	0
STRUTS	1.1.0	1.2.4	123	6	1	14	109	43	80	29	69	36	82	28	84	7	0
STRUTS	1.2.4	1.2.6	2	0	0	0	2	0	2	0	2	0	2	0	2	0	0

equivalent functionality between two methods, although it proves slightly less precise than using the edit distance. However, in a number of cases, binary incompatible methods did not have a code comment associated with them; thus while all lexical techniques had a roughly similar precision, techniques based on code comments were able to recover only a small number of changes.

The deprecation inference technique performed particularly poorly, worse than previous work suggested (see Section 5.3). I found that it was difficult to reliably extract recommended replacements from code comments in these migrations, due to the high degree of variability in how the comments may appear. In many cases, a name of a replacement method is simply mentioned in the comment body, but without syntactic cues to indicate that it refers to a method (e.g., a fully qualified type name, parentheses following an identifier to indicate that it refers to a method or function). This should not be taken as an indicator that this inference technique is not useful, but rather that automatically extracting this information for these migrations is challenging. I report the results for deprecation comments here for completeness, but I do not incorporate this inference technique elsewhere in the paper.

7.2 Flexible Hybridization

Individually, the inference techniques examined in Section 7.1 are not able to resolve many of the binary incompatible changes to API methods found in these migrations. However, each of these techniques leverages different characteristics of the method's implementation to find related functionality, such that in cases where one technique fails another could be viable. Previous work has already suggested that many of these approaches have a degree of orthogonality (see Section 5.3 and Section 6.5), and thus could be combined in some fashion to overcome each technique's limitations in a collective fashion. However, given that each technique's effectiveness is variable, and new inference techniques may be proposed in the future that could be more effective than these, how can they be combined to take advantage of their best results, without needing to have perfect knowledge of how they *should* be combined?

7.2.1 Challenges in Hybridization

Previous attempts to hybridize inference techniques into a composite change recommender for API migration have relied on tool creators to understand the strengths and weaknesses of a particular technique, tailoring the combination of inference techniques in highly specialized, predetermined ways. For example, in Section 7.1.2 the results for call analysis show it to have few false-positive predictions, but because its effectiveness relies on finding usage examples for the affected portions of the API in the library's implementation, it is limited in the contexts where it can recommend changes. Consequently, it is often hybridized with additional techniques aimed at expanding its results set, such as adding additional sources of usage examples from external implementations [Schäfer et al., 2008] or from the library's revision history [Dagenais and Robillard, 2011], or using other inference techniques to resolve those changes upon which call analysis has no data to operate [Wu et al., 2010].

However, these combinations are not done arbitrarily, nor easily: integrating these additional sources of data or inference techniques is typically done in a tightly coupled fashion such that each addition often can require extensive reworking of the change recommender's implementation. For instance, the incorporation of a library's revision history into the set of source code that call analysis can glean for usage examples of affected API entities requires the integration of an additional analysis technique for resolving the binding information from the individual fragments of JAVA code that are incrementally checked in over the library's development cycle [Dagenais and Robillard, 2011]. This is clearly non-trivial to add, but can also complicate the addition of other techniques into the change recommender.

Furthermore, as Section 7.1.2 shows, the effectiveness and appropriateness of each inference technique can vary wildly even within the same API. A combination of techniques that proves effective for one API may be less effective for others, or could benefit from the incorporation of another inference technique that is more promising in that domain.

7.2.2 Machine Learning for Change Recommenders

I see value in constructing hybrid change recommenders, in which the combination of inference techniques is flexible; that is, the number and kinds of inference techniques in use can be easily adjusted, perhaps even by the developer actually using the recommender for their work. Rather than relying on heavily customized interactions between the inference techniques, I would like to instead use data either from past migrations or from developer feedback on predictions, to determine how to recognize when the feedback from a set of inference techniques suggests a correct replacement. I would also like to have the option of adjusting the kinds of inference techniques in use to take advantage of new techniques, or remove techniques that are not making a difference.

I borrow from the idea of "boosting", a machine learning technique in which a series of weaker classifiers are combined into a stronger classifier through the application of training data [Freund and Schapire, 1997]. I chose to evaluate the effectiveness of three machine-learning techniques in combining the recommendations of inference techniques into a new hybrid change recommender, where the outputs of all inference techniques are collated, and then reanalyzed to determine if the aggregate similarity ratings suggest a particular replacement is valid. I evaluated two rule-learning techniques, as my impression of the data suggested that rule-based approaches might fare well. I also used a naive Bayesian recommender technique, as they are widely applied and often suited to recommendation problems in general [Jannach et al., 2010].

1. *Naive Bayesian recommenders* [John and Langley, 1995; Jannach et al., 2010] use training data to calculate the probabilities that (in this case) a given set of recommendations and similarity values suggest a correct or incorrect match of functionality; these probabilities train the actual (automated) learner.

2. *Decision tables* [Kohavi, 1995], a tabular form of a decision tree, use training data to elicit a set of rules that best governs the data, refines the rule set by eliminating attributes (i.e., inference techniques) that have small or negative impact, and then uses the table to effectively lookup a classification based on the ratings of each

attribute.

3. *Rule-induction approaches* [Cohen, 1995; Jannach et al., 2010] examine a portion of the training data to incrementally learn rules governing the characteristics of correct matches. I used the RIPPER algorithm of Cohen [1995].

7.3 Case Study

To evaluate the comparative effectiveness of these machine learning techniques, I built a testbed hybrid change recommender which, given the source code of two library versions and a list of binary incompatibilities introduced into the library API versions, applies each of the inference techniques detailed in Section 7.1 to the source code to obtain a set of recommended replacement functionality for each binary incompatible method-level change. The testbed then aggregates and correlates these recommendations so that, for each BI, a list of recommendations is created, listing the similarity ratings for each inference technique for that recommended change. Using the data from the Corpus, I then mark recommendations as correct or incorrect, and the result is a corpus of data with all predictions from all techniques, as well as whether the outcome of those predictions are right or wrong.

As in Section 7.1, I evaluated these techniques on the same libraries from the Corpus, as well as the LUCENE library. However, to provide a fair comparison to the work of Dig et al. [2006], Schäfer et al. [2008], and Wu et al. [2010] on hybridized approaches, I restricted the study to consider only 1:1 method-level BI changes in each migration.

7.3.1 Methodology

I used standard implementations (provided by the WEKA library for ML [Hall et al., 2009]) for each of the ML techniques in Section 7.2.2 to configure a hybrid change recommender using the aggregate inference data from each of the techniques. The inputs for each machine learner are a specific recommended replacement for an API change, and a set of similarity ratings generated

from each of the applied inference techniques; the machine learner is then asked to indicate if this replacement is correct or incorrect, given the supplied similarities.

I perform 10-fold cross-validation [Stone, 1974] on the dataset: the data is randomly divided into 10 subsets of equal size: 9 subsets are chosen as the training set with the 10th used as the test set. The hybrid recommender is trained using the training subsets, and then evaluated using the test dataset to determine if a recommendation is correct, given its knowledge of which similarity measures are more likely to successfully predict a correct replacement. This judgment is compared to the known results, and the correctness of the recommendation for that replacement is recorded. This process is repeated 10 times, by varying which of the 10 subsets is chosen to be used as the test set, and using the remaining 9 subsets for training. The number of correct and incorrect recommendations recorded are the average across the 10 trials.

I recorded correctness as follows: correct recommendations are marked as true positive matches; wrong recommendations are marked as false positives; and a lack of a recommendation when such a recommendation would be correct is marked as a false negative. I do not record true negatives (i.e., cases where a recommendation is marked as wrong by the ML technique, and for which it is known to be wrong) due to their large numbers.

7.3.2 Results

Table 7.2 presents the accuracy of the trained hybrid recommender using each of the three machine learning techniques detailed in Section 7.2.2, broken down by library API. The *precision* of each hybrid recommender indicates the percentage of correct recommendations for the library migrations involved from all of those recommendations that were made; the *recall* reflects the percentage of all the correct adaptations for those libraries in the test data set that were recommended.

For each of the recommenders, I show its effectiveness when only a subset of the available inference techniques are in use, and I demonstrate the improvements or degradations in accuracy as additional sources of data are added. For all techniques, I begin with using the data from call analysis and structural similarity; the choice here reflects the relatively high precision of call analysis,

Table 7.2: Comparison of ML technique accuracy, as percentages. CA = call analysis, SS = structural similarity, Sig = LED and LCS analysis of method signatures, Com = LED and LCS comparison of comment text for affected methods.

Technique	API	CA+SS		CA+SS+Sig		CA+SS+Sig+Com	
		precision	recall	precision	recall	precision	recall
Naive Bayesian	JDOM	34.4	78.6	36.6	66.7	35.4	47.2
	LOG4J	71.4	100.0	80.4	95.3	59.7	65.2
	LUCENE	21.1	66.7	26.9	43.8	25.9	41.2
	STRUTS	3.6	75.0	3.6	75.0	10.9	50.0
Decision table	JDOM	26.7	88.99	30.3	71.4	28.6	78.3
	LOG4J	73.5	100	64.7	94.3	64.7	94.3
	LUCENE	65	81.3	57.1	75.0	57.1	75
	STRUTS	15.7	92.9	37.8	90.0	28.7	83.3
Rule induction	JDOM	35.6	95.5	34.9	81.5	32.8	87.0
	LOG4J	70.0	100.0	80.0	97.6	72.5	94.9
	LUCENE	65.0	81.3	65.0	81.3	65.0	81.3
	STRUTS	15.7	92.9	44.9	85.1	47.8	84.3

and some previous success with structural similarity [Cossette et al., 2014]. I then incrementally add inference techniques, starting with the two lexical techniques (edit distance and longest common substring) to compare method signature similarity, and then the data on comment similarity using the same two lexical techniques. The precision and recall of the trained hybrid are shown for each combination.

There are two interesting effects that I note in Table 7.2. First, while each of the ML techniques works fairly well on changes to the JDOM and LOG4J libraries, the naive Bayesian recommender struggles significantly with changes to the LUCENE and STRUTS libraries. This reflects the fact that the inference techniques used on these libraries (Table 7.1) have a high number of incorrect predictions mixed in with the correct predictions for these cases, contributing to a substantial degree of noise in the data. As a result, there are several inconsistencies in the data: a similarity value that is a correct prediction in one case, may be incorrect in another. The rule learners seem to fare

better in these cases, particularly the RIPPER rule-induction approach, as it bases its rule set on correct examples from the training data, rather than all examples. As a consequence it seems to fare better when inconsistencies appear in a given technique's predictions, and seems to be is less susceptible to the influence of a large number of wrong examples.

The second effect is a clear indication that the addition of an inference technique's analysis does make a contribution, but not always, and not for every library. For example, the addition of techniques using lexical analysis on method signatures often improved the overall precision of the hybrid recommender, but usually at the expense of a drop in the hybrid's recall; in some cases, the addition offers little if any benefit. More interesting is the effect of analysis on code comments: for some APIs, the similarity of altered method comments actually confuses the hybrid recommender, reduce the precision and recall of its recommendations; for others, it offers small improvements for other migrations it offers a small improvement. This suggests that the configuration of inference techniques used to assist in a library migration should not always be predetermined; in some cases, a developer may notice that their results improve with additional techniques, and in others they may find degradation.

Table 7.3: Comparison of ML technique accuracy across all APIs, as percentages.

Technique	CA+SS		CA+SS+Sig		CA+SS+Sig+Com	
	precision	recall	precision	recall	precision	recall
Naive Bayesian	0.0	0.0	1.4	27.3	18.6	46.2
Decision table	38.1	95.2	41.7	90.0	42.1	90.1
Rule induction	37.6	95.2	49.3	87.9	52.0	87.2

Table 7.3 presents a second analysis where the recommendation data from all migrations and APIs is aggregated into one dataset. 10-fold cross-validation is again used. For the rule-based techniques (decision tables and RIPPER), the results show an aggregate effectiveness that is better than the individual techniques. While the overall effectiveness is poorer than for some individual migrations, it is still quite effective, and is suggestive that with sufficient training data, these hybrid

Table 7.4: Comparison to REFACTORINGCRAWLER, Schäfer et al., and AURA for 1:1 method-level BI changes in STRUTS 1.1.0 to 1.2.4.

	Naive Bayesian	Decision Table	Rule Induction	REFACTORING CRAWLER	Schäfer et al.	AURA
True Positive	36	34	41	19	7	27
False Positive	128	7	15	0	0	41
False Negative	17	19	12	34	46	26
Precision	22%	83%	73%	100%	100%	40%
Recall	68%	64%	78%	36%	13%	51%
F-score	33%	72%	75%	53%	23%	45%

recommenders could be more broadly useful on libraries. However, the naive Bayesian approach fares significantly worse; again, the large number of wrong results in the dataset, though appropriately classified, seems to overwhelm the approach and cause it to fare poorly in most cases.

Finally, Table 7.4 shows the effectiveness of the hybrid recommenders on the STRUTS 1.1.0 to 1.2.4 transition, compared to three change recommenders that were also evaluated on this transition: RefactoringCrawler [Dig et al., 2006], Schäfer et al. [Jonas, 2007], and AURA [Wu et al., 2010].[9] A total of 53 method-level, binary incompatible changes whose replacement was a 1:1 transformation are present in this migration. The true positive, false positive, false negative, and accuracies of each techniques are calculated from the prediction data each author provided, by comparing predicted replacements to the list of known BI changes recorded in the Corpus. Wrong predictions were re-investigated to determine if it instead represented an equally valid substitution or an error in the Corpus (see also Appendix E). Because methods added to interfaces, and methods belonging to class-level BIs are not considered as in Section 6.5, the accuracy of the comparison techniques is higher in this evaluation.

[9] As in Section 6.5, a comparison with the HIMA tool of Meng et al. was not done as I was unaware of that work at the time of this research. A comparison of the HIMA tool's effectiveness to these techniques can be found in Appendix D.

7.3.3 RQ 7.2: Could end-users successfully configure hybrid combinations?

The results suggest that given training information by an end-user to configure a machine-learning based hybridization of inference techniques, an API change recommender can be constructed that successfully leverages the strengths of the applied inference techniques. Particularly, the results show that decision tables and rule induction hybrids are significantly more effective at correctly recommend replacement functionality than either the Bayesian machine-learning hybrid, or the compared techniques of Schäfer et al., REFACTORINGCRAWLER, or AURA. For the Struts migration, the Bayesian learner is prone multiple recommendations for every method-level BI, with most of them proving to be wrong. By comparison, the decision table and rule induction hybrids fair better at recognizing errant predictions from their component analysis techniques.

When compared against the more favourable testing conditions for Schäfer et al., REFACTORINGCRAWLER, or AURA, rule-based hybrids still prove to be more effective. While some techniques show higher precision through overly restrictive analysis, the F-score (which is the weighted average of precision and recall) shows the hybrid approaches to be significantly more accurate in recommending replacement functionality. Overall, the Rule Induction fared the best, with an F-score of 75%.

7.4 Discussion

There are threats to the internal validity of my work. First, the inference and machine learning techniques I have selected for this study may be influenced by both my previous experience with empirically evaluating these techniques, and my knowledge of the libraries used in the studies. However, all of these inference techniques have been used by one or more change recommenders proposed in the literature, and were picked because of their commonness in this area; furthermore, the effectiveness of these techniques as reported in Section 7.1 shows that some fare poorly on certain migrations. The choice to consider two different rule-learners does reflect my opinion that these may fare well with noisy datasets, but I do not believe these are the only techniques that could

work well in this task; other work suggests that in many cases, it is difficult to know if a particular machine learning approach is suitable for a problem area until it is applied [Jannach et al., 2010]. I actively hope that others in the community will investigate other machine learning techniques in this area, and improve on these results.

My work has two threats to external validity, specifically the generalizability of these results. First, this work has evaluated the effectiveness of these inference techniques on the method-level binary incompatible changes of four JAVA APIs; the study of the effectiveness of the applied machine learning techniques for hybrid recommenders was further restricted to consider only 1:1 changes. I recognize that these libraries, and these API change characteristics are not reflective of all libraries, and thus there may be significant variation in the results of any of these approaches when applied to other systems. That said, the results are promising, suggesting that hybrid approaches that support flexible combinations of inference techniques can be effective in recommending migrations for some APIs. Furthermore, I consider the methodology applied to itself be a contribution, illustrating the problems of any fixed configuration for a hybrid, which has been the norm to-date.

Secondly, the evaluation does not address the cold-start problem [Jannach et al., 2010]: namely, the problem of applying the recommender when no examples are available for training prior to use. In this scenario, the developer would likely identify correct replacements from the recommendations provided by the underlying techniques, thus training the hybrid recommender over time until it was useful. While the evaluation shows these hybridizations are far more effective, it does speak to the technique's performance when limited training data is available. A first step towards understanding this problem would to apply cross-fold validation as before to train and test the machine learning algorithms, but starting with 10% of the data allocated to training (rather than 90%), and progressively evaluating the tool's effectiveness as incrementally larger portions of the data are allocated to training, thus simulating the usage of the tool over time.

7.5 Summary: Hybridizing Recommenders for Reactive Change

I have presented an evaluation of how flexible hybrid API change recommendation systems can be constructed, using a combination of existing API change inference techniques, training data, and rule-based machine learning techniques. The evaluation shows that, for the APIs under examination, the effectiveness of API recommenders can be improved by combining the results of other inference techniques, and using training data to guide the construction of rules that help determine when the similarities provided by several inference techniques are suggestive of a correct adaptive change. Furthermore, my work shows that there are cases in which adding additional inference techniques can help or hinder the success of such tool support, and thus the developer may be helped by having the option to add or remove analysis techniques as they see fit to improve the quality of recommendations they receive.

Chapter 8

Discussion

The appropriateness of semi-automated approaches for supporting change impact analysis and redress reflects the difficulty of the problems in this area. Full automation requires a predetermination of how the tool support should be configured: that is, the analyses applied must be decided in advance, and the meta-analysis of their individual results must also be predetermined to arrive at a single recommendation. This proves problematic, as the complexities of change impact analysis in polylingual systems, or redress in library API evolution, arise from the varying characteristics possible in any particular software system.

The presentation on how libraries evolve, and how to appropriately recommend adaptations to react to their API changes in Chapters 5, 6, and 7 reflect this. At the outset of my investigation into API evolution, existing work at the time strongly suggested the vast majority of API changes are caused by refactorings (see Section 3.4.1), and were thus ideally supported by automated approaches for identifying and automating the migration of refactorings (see Section 3.3.1 and Section 3.3.3).

However, the empirical study of Chapter 5 found that for the libraries analyzed, most API changes had complex causes that are difficult for the applied techniques to resolve, and whose nature *required* developer input to migrate code dependent on that API change to the new version (see Section 5.2). Further, a given analysis technique proved more effective (sometimes three times as much) in identifying adaptive changes for one library as opposed to others for which its analysis was less suited for the library, or the changes it underwent. Predetermined analysis combinations ignore the reality that every library's implementation and change characteristics are different, and may not match those the tool developer had in mind when constructing analysis support. Relying on a developer instead to provide examples for configuring tool support can thus identifying the

analyses that are more effective on their software systems (see Section 7.3).

This was first recognized in my work on polylingual dependency analysis for change impact prediction (see Chapter 4). The language and technology combinations possible made automating a general-purpose approach for dependency analysis impossible; targeting tool-support for a specific set of languages and technologies is expensive, and previous work has shown that adapting to other languages or technologies is extremely difficult (see Section 3.1.2). The alternative, a manual application of lexical-aware tools, was cumbersome and prone to spurious matches.

What proved successful was providing a means by which developers could configure tool support by describing the necessary syntax to capture dependency relationships in their software system's context (see Section 4.5). The developer is assisted in configuring the tool by its focus on dependency analysis (rather than more general purpose reverse engineering analyses), allowing a tailoring of the notation for specifying dependency syntax. A useful, semi-automated solution becomes available then to developers in contexts where generic, fully-automated approaches does not exist, and where largely manual analyses are ineffective.

8.1 The Library Migration Corpus

A key problem I faced throughout this work was finding useful data to guide my analysis and verify the results of tools; it was necessary for me to determine through manual investigation where dependencies existed (see Section 4.5.1), or how a library had changed (see Section 5.1 and Section 6.5). This has consequently limited the generalizability of these findings.

While I have discussed these limitations before (see Section 4.7, Section 5.5, Section 6.6, and Section 7.4), the broader issue is worth discussing further: the lack of publicly available, empirical data describing how software systems change limits research, and hampers understanding the effectiveness of new approaches. Dependency analysis in polylingual systems, as well as library evolution and API migration, are notable examples of this: tools were of limited effectiveness, requiring a largely manual approach to constructing a complete dataset.

As part of this dissertation, I have started to address this problem through the creation of "The Library Migration Corpus" ("the Corpus"). Using the data I collected in Section 5.2 and Section 6.5 to understand how JAVA libraries changed, I created a publicly available set of data that completely describes the binary incompatible changes made to the STRUTS, LOG4J, JDOM, LUCENE, SLF4J, and DBCP libraries over multiple releases (see Section 5.1 and Section 6.5). The research community is invited to download and use this data in evaluating their own tools, provide corrections to the data, and submit additional sources of library migration data to be hosted with the Corpus. While the generalizability of my results are limited today by the number of libraries evaluated on, I hope that in establishing this Corpus it will eventually become a community undertaking. As other researchers contribute in-kind to the corpus with library change data for other systems, all will benefit from having access to a growing body of complete library change data.

The Library Migration Corpus has an opportunity to provide useful feedback and guidance to researchers interested in establishing a taxonomy of changes, identifying additional change metrics for studying library evolution, and a robust set of ground-truth data for objectively evaluating new ideas and approaches in this area. It is currently available at http://lsmr.cpsc.ucalgary.ca/projects/corpus, as well as at lsmr.org. The Corpus is also provided in Appendix E.

8.2 Future Work

There are several avenues for further exploration of this work, two of which I will discuss here.

8.2.1 Case Studies of Semi-Automated Redress for API Migration

In Chapter 7, I showed that machine learning techniques could be viably trained through supplied examples to recognize when a set of API change recommendation techniques correctly recommended an adaptation for a given API change. The next step will be a concrete realization of this approach in a prototype suitable for industrial evaluation. Important questions that need to be answered are: (1) how many examples does a developer need to provide until a tool is sufficiently

useful; (2) where can useful examples come from (e.g., only the current migration, previous migrations, etc.); and, (3) does an iterative refinement of an API recommender configuration fit into the workflow of a developer undertaking an API migration task? While the current vision of this tool involves having developers choose which analysis techniques are to be applied, followed by a manual specification of examples, there may be opportunities to passively observe changes a developer makes to code dependent on an API, and use these as examples for suggesting how hybrid tool support can be configured, such as: (a) suggesting the inclusion of analysis techniques most suited to identifying changes the developer has been observed making, as they may indicate those techniques that are suitable on this software context; and, (b) asking if the developer's modifications are worth using as training examples for configuring hybrid tool support. There are interesting usability challenges in realizing this approach, particularly in providing prediction traceability to developers that shows why a given recommendations is made, and whether this improves developer confidence in the results.

Later work could seek to adapt a set of API change recommendations into a semi-automated migration plan, adapting formal migration and other source code transformation techniques (e.g., [Chow and Notkin, 1996]) to the needs and capabilities of industrial software developers. As with the DSKETCH tool's "Sketch" notation in Section 4.1, it may be possible to simplify the specification of an API migration, by using the catalogue of transformations identified in Section 5.4 to inform an appropriate notation. Ideally, commonly occurring transformations should be easy to specify, and special cases noted in that chapter that would otherwise necessitate detailed specifications could be simplified appropriately (e.g., noting that an eliminated sub-type had no fields or method declarations, and thus is effectively equivalent to its parent, should be easier to specify than a complete detailing of how they are equivalent). It would also be desirable for such tool support to be integrated with API change recommendation tooling, such that when a developer identifies an adaptive change, the tool can record the needed transformation (similar to the CATCHUP! tool of Henkel and Diwan [2005], and the GILLIGAN tool of Holmes and Walker [2007]), asking the

developer for additional details if the transformation is complex.[1]

Finally, in keeping with the spirit of semi-automated support, such migration support should not require a complete specification to be useful. Instead, it should allow the provision of "just enough" details to enact most of the transformation for a migration if desired, for cases where manual enactment of the final changes proves easier than specification. In my evaluation of DSKETCH in Section 4.5, a participant configured DSKETCH to find most of the dependencies for their change impact analysis, but stopped when they realized that a small amount of additional manual investigation was sufficient to complete the task. Similar capabilities could be desirable in migration tool support, particularly if certain migration changes are involved and localized to small regions of the code, where writing a detailed specification may be more work than enacting the change.

8.2.2 Technical Risk Estimation in API Migration

One of the ongoing challenges in choosing whether (or when) to migrate software to use a newer version of a library is that the costs of the upgrade are unknown. Documentation of a new library release often touts the benefits offered in that new version, including new functionality, performance improvements, and bug fixes; but as shown in Section 5.2, most of the breaking changes made to the API are undocumented. The consequence is that while the *benefits* of upgrading to a new library version are typically known, *the risks and costs* of undertaking the migration are not: as seemingly innocuous changes to the API can require complex adaptations to dependent code, the true cost of a migration is often only known when it is finally complete.

It cannot be the case that a developer *must* always migrate their software to use the latest library version; for some library versions, the risks of additional work and unanticipated costs occurring due to library migration, will outweigh the benefit of the upgrade itself. Ideally, identifying this *technical risk* in library migration should be determined in advance, and presented to the technical leadership in charge of the software's development strategy: they could then decide if the risks

[1] In effect, the support would adopt the PERL maxim of "Make easy things easy, and hard things possible." [Wall and Schwartz, 1991]

were acceptable in light of the purported benefit.

An interesting possibility here would be to apply *speculative analysis* [Brun et al., 2010; Muşlu et al., 2012] to the predictions of a hybrid API recommender technique. Speculative analysis was originally developed to suggest possible repairs to developer code, and was used to apply a series of likely fixes to a compilation error to see what the resulting outcome was, and to suggest to a developer how they should correct their code. I see a more sophisticated application of the idea here, using the proposals of an API change recommender to seed a pseudo migration plan for a proposed library upgrade. For each binary incompatibility detected in the new version that impacts the dependent software system, an API change recommender will provide one or more adaptations to the speculative analysis engine. The analysis engine will then attempt to apply the migration to the dependent software system, through one of two ways:

1. the adaptation is applied to a clone of the source code (e.g., through a process like *twinning* [Nita and Notkin, 2010]), which is then recompiled, and resulting compilation errors are noted; or,

2. for complex changes, the structure of the adaptation is anti-unified against the original code to suggest the points in common, and the points at which the adaptation differs from the implementation, which can then be used to estimate the difficulty of enactment (see Chapter 6 and Cottrell et al. [2008]).

An initial application of such tooling could provide a workable first estimate of the technical risk of the change, along with an initial proposed migration plan. This estimate could then be iteratively refined by a developer by correcting or modifying predicted adaptations, or adjusting estimated risks for a given change, until the developer and their technical leadership felt the plan and its technical risk estimate was sufficiently detailed to make an informed decision.

Such work could be expanded to simplify the derivation of technical risk estimates, through identifying constraints in the API or the source code that imply a proposed adaptation is unworkable, or through empirical cost investigations as to the relationship between the "distance" of a recommended solution from the dependent source code, the developer effort necessary to enact it.

This may provide opportunities to estimate technical risk in other software evolution domains, perhaps by using a system's change history, correlated with task descriptions or bug reports, to suggest correlations between key words in a change or task description, and the resulting risks in enactment. A combination of speculative analysis and anti-unification of previous change histories against the current software state may suggest possible evolutionary paths a system may undergo, that could be mapped to such key words, perhaps one day allowing a business analyst with a non-technical background to eventually see estimates based on past development history of the costs involved in a modification they propose, due to the similarity of features they request to those enacted in the past.

8.3 Summary: Discussion

Semi-automated approaches are useful for addressing problems of polylingual dependency analysis, and library API migration. Because change impact analysis and redress in these contexts must cope with additional complexity introduced by language and technology combinations, or the nature of enacted changes, fully automating solutions requires extensive analysis capabilities that are expensive to implement, and tailored to specific problems that are not representative. Semi-automated approaches that rely on developer input to resolve complex problems can dramatically improve the effectiveness of analysis in these areas.

As part of my work, I have also established the first Library Migration Corpus, comprising a complete set of library change data for six JAVA libraries over multiple versions. The Corpus provides a ground-truth understanding of how libraries and their APIs evolve, as well as independent *a priori* knowledge of library changes useful for the complete and thorough evaluation of API

change recommendation and migration techniques. This Corpus is available to the public, and is actively encouraging usage and contributions from the community at large.[2]

My work has several near-term and long-term applications that I yet to explore. Having identified a framework for the flexible hybridization of API change recommenders, based on developer-provided migration examples, it remains to be seen what challenges this approach would face in practice. Promising augmentations include incorporating the opportunity to save correctly identified migrations into a transformation specification, that developers could later use to migrate their software to a new API version. This could be combined with a simplified means for specifying API transformations, based on the characteristics of change categories my research identified as requiring minimal additional specification in some circumstances. Such a recommender and migration framework could support applications in assessing the technical risk associated with library migrations prior to their undertaking, allowing developers to work in concert with non-technical leadership to decide when the costs of a library migration are justified by their benefits.

[2]The Library Migration Corpus is available at http://lsmr.cpsc.ucalgary.ca/projects/corpus, as well as through the new *Laboratory for Software Modification Research (LSMR)* home page: http://lsmr.org

Chapter 9

Conclusions

Software systems must change over time if they are to remain useful and relevant to their users, but such changes carry the risk of negatively impacting a software system. While a variety of approaches aim to support developers in change impact analysis and redress tasks, the support needed depends on whether their work is proactive, or reactive. Proactive investigation of dependencies for change impact analysis is supported through automated tools, but these prove expensive to construct for every programming language, and thus are ill suited for polylingual systems. Reactive redress of change impacts from external library evolution is also supported by change recommendation tools, but my work shows such tools address a small subset of the changes that occur between API versions. Automated approaches in these contexts must be tailored to specific problem domains to be effective, and in so doing cannot generalize to other similar systems, tying their implementation costs to one specific application.

The thesis of this work is that semi-automated approaches to change impact analysis and change recommendation systems provide more effective tool support than fully-automated or manual approaches. Semi-automated approaches rely on developers using the tool support to provide key details on their problem domain, or feedback on the tool's predictions, to iteratively refine a tool's analysis to fit their software's characteristics. As a consequence, semi-automated approaches provide analysis capabilities that prove correct in a wider array of software systems and contexts, and thus prove more effective in addressing the problem of change impact and redress. I have conducted two lines of research to provide evidence to support this thesis.

My work began with an investigation on supporting dependency analysis in polylingual systems by the lightweight specification of a simplified "island grammar" to capture dependency syntax in source code, with the intent that such specifications by easily learned by industrial developers. A

collection of these specifications for each language in a software system would then provide the basis for generating static dependency analysis tooling for cross-referencing polylingual dependencies in the software; developers would then iteratively refine their specifications to improve the tool's effectiveness. A reification of this approach in DSKETCH tool was provided to industrial developers for a case study in which they needed to determine dependencies for modification tasks in open-source polylingual systems. Participants proved able to learn the "Sketch" notation with minimal training, were able to reuse and extend portions of specifications provided by other developers when appropriate for the languages and technologies in their systems. Participants discovered dependencies using DSKETCH they previous missed using other available analyses, and noted the increase in tool effectiveness lead to greater trust in using the tool.

I continued by studying the evolution of software libraries and their APIs, do determine the nature of breaking changes introduced between API versions, and their implications for redressing their impacts on dependent code. I conducted a detailed, largely manual investigation of the binary incompatibilities between pairs of library versions, and constructed the first publicly available "Library Migration Corpus" to detail breaking changes to libraries, as well as the necessary adaptive changes. In the course of my investigation, I found that existing change recommendation techniques were generally unsuccessful in identifying correct replacements, and proved on average to be successful for only 20% of the binary incompatibilities encountered. Further, the majority of API changes I observed could not be automatically migrated without additional specifications provided by a developer, suggesting that automated migration approaches are of limited use.

An important outcome of this work was the observation that individual change recommendation analysis techniques showed some degree of orthogonality; the majority of binary incompatible API changes were resolvable by only one or two such techniques. This suggested that: (i) additional analysis techniques were necessary, as existing approaches provide insufficient coverage; and, (ii) a collection of such techniques may ultimately prove necessary to identify the adaptive changes needed for redressing the impacts of library API change.

To expand the collection of available analysis techniques for API change recommendation, I proposed using matching via structural generalization (MSG) to detect correspondences between the syntactic structures in a library's API and its implementation, suggesting similar or equivalent functionality between library versions that may have undergone lexical or minor syntactic modification. Reified in the UMAMI tool, it proved able to correctly determine adaptations for 56% of BI methods, and proved significantly more effective when compared to other API change recommender techniques. By itself, matching via structural generalization is insufficient to resolve the all binary incompatible changes in the libraries analyzed; however, as with the techniques evaluated before, MSG proved capable of discovering replacement functionality for changes where other techniques are unsuccessful. MSG in concert with other analysis techniques could provide a much more effective approach for identifying API change recommendations.

Combining API change analysis techniques for hybrid recommendation requires flexibility: my empirical work showed each technique's effectiveness varied significantly when applied to different libraries. A fixed combination of techniques, with a predetermined meta-analysis of their results to determine which is correct, cannot effectively analyze some libraries. Instead, I proceed from the assumption that a developer can be relied upon to provide examples of correct migration adaptations from either previous API migrations, or the current migration. These examples can then be analyzed by a set of change recommendation approaches, and their results used to train a machine-learning approach to recognize the circumstances in which one or more analysis techniques is correct. In so doing, the machine-learner develops a hybrid meta-analysis of change recommenders that reflects the particular combination of techniques in use, and the characteristics of the library on which it is trained. My evaluation of three different machine-learning approaches showed that example-trained, rule-based machine learners were able to find the correct replacement functionality for binary incompatible changes made to libraries in up to 94% of cases. A comparison of this flexible approach to existing API change recommenders showed dramatically higher F-score measures of accuracy, improving on comparable approaches by over 20%.

Semi-automated approaches lend themselves well to the creation of tool support for addressing problems of polylingual dependency analysis, and library API migration. Because change impact analysis and redress involves complex software evolution problems, fully automating solutions requires extensive analysis capabilities that are both expensive to implement, and limited in their application. However, relying on developer input at key points during the analysis of these problems can dramatically reduce the difficulty in their analysis, and increase the likelihood of success. By preferring input and feedback over extensive analysis, and allowing the developer to dictate when a solution is appropriate, semi-automated approaches can allow for the creation of tailored solutions that prove more effective than fully automated or manual alternatives.

Proactive and reactive responses to change impact and redress in software systems need to cope with the increasingly complex and specialized nature of the changes developers must resolve in responding to the consequences of software evolution. By relying on developer input to resolve difficulties, rather than requiring extensive analytic implementations with limited application, it is possible to provide analytical tools for developers working areas that proved too difficult to support in the past.

9.1 Contributions

There are seven major contributions of this dissertation.

1. A novel approach to lightweight tool support for polylingual dependency analysis, which leverages simplified developer specifications to generate customized tooling for dependency analysis (see Chapter 4).

2. The first systematic classification and analysis of the source code transformations necessary to adapt to library change (see Chapter 5).

3. The first corpus of complete library change data, for use in evaluating and replicating library migration techniques (see Chapter 5 and Appendix E).

4. A tool-independent comparison of the effectiveness of change recommendation techniques in resolving all changes to a software library's API (see Chapter 5 and Chapter 6).

5. A new inference technique for API version migration, *matching via structural generalization*, and an empirical analysis of its strengths and weaknesses relative to both existing change recommenders and current analysis techniques (see Chapter 6).

6. A general framework for hybridizing inference techniques for change recommendation in API migration, and an empirical evaluation on the effectiveness of several configuration practices (see Chapter 7).

7. A detailed analysis of the published results for existing API change recommender tools, comparing the correctness of tool predictions to those established independently in *a priori* change data (see Appendix D).

Bibliography

Jesper Andersen and Julia L. Lawall. Generic patch inference. *Automated Software Engineering: An International Journal*, 17(2):119–148, June 2010. doi: 10.1007/s10515-010-0062-z. 2, 63

R. S. Arnold and S. A. Bohner. Impact analysis: Towards a framework for comparison. In *Proceedings of the Conference on Software Maintenance*, pages 292–301, 1993. doi: 10.1109/ICSM.1993.366933. 2, 3, 22, 24, 43

D. C. Atkinson and W. G. Griswold. Effective pattern matching of source code using abstract syntax patterns. *Software: Practice & Experience*, 36(4):413–447, 10 April 2006. doi: 10.1002/spe.704. 4, 29, 35

Ittai Balaban, Frank Tip, and Robert Fuhrer. Refactoring support for class library migration. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 265–279, 2005. doi: 10.1145/1094811.1094832. 5, 56, 60, 62, 64, 113

Friedrich L. Bauer. Software engineering. In *Proceedings of the IFIP World Congress*, volume 1, pages 530–538, 1971. 20

Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2nd edition, 2004. 11

L. A. Belady and M. M. Lehman. A model of large program development. *IBM Systems Journal*, 15(3):225–252, September 1976. doi: 10.1147/sj.153.0225. 1, 21

Ted J. Biggerstaff. The library scaling problem and the limits of concrete component reuse. In *Proceedings of the International Conference on Software Reuse*, pages 102–109, 1994. doi: 10.1109/ICSR.1994.365806. 3

Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas E. Webster. Program understanding and the concept assignment problem. *Communications of the ACM*, 37(5):72–82, May 1994. doi: 10.1145/175290.175300. 4, 32, 67

Walter R. Bischofberger. Sniff: A pragmatic approach to a C++ programming environment. In *Proceedings of the USENIX C++ Technical Conference*, pages 67–82, 1992. 39, 40

Joshua Bloch. How to design a good API and why it matters. In *Companion to the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 506–507, 2006. doi: 10.1145/1176617.1176622. 44, 45, 93, 245

S. R. Bourne. *An Introduction to the UNIX Shell*. Bell Laboratories, 1977. 29

Frederick P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987. doi: 10.1109/MC.1987.1663532. 1, 2, 3, 21

Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Speculative analysis: exploring future development states of software. In *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research*, pages 59–64, 2010. 185

Peter Bulychev and Marius Minea. Duplicate code detection using anti-unification. In *Proceedings of the Spring Young Researchers Colloquium on Software Engineering*, pages 51–54, 2008. URL http://www-cad.eecs.berkeley.edu/~marius/papers/syrcose08.pdf. 131

Peter Bulychev and Marius Minea. An evaluation of duplicate code detection using anti-unification. In *Proceedings of the International Workshop on Software Clones*, 2009. URL http://www.informatik.uni-bremen.de/st/IWSC/bulychev.pdf. 131

J. Burghardt. E-generalization using grammars. *Artificial Intelligence Journal*, 165(1):1–35, June 2005. doi: 10.1016/j.artint.2005.01.008. 130, 131

S. Burson, G. B. Kotik, and L. Z. Markosian. A program transformation approach to automating software re-engineering. In *Proceedings of the IEEE International Computer Software and Applications Conference*, pages 314–322, 1990. 31, 67

Yih-Farn Chen, Michael Y. Nishimoto, and C. V. Ramamoorthy. The C information abstraction system. *IEEE Transactions on Software Engineering*, 16(3):325–334, March 1990. doi: 10.1109/32.48940. 3, 4, 25, 26, 31, 32, 70

Elliot J. Chikofsky and James H. Cross, II. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, January 1990. doi: 10.1109/52.43044. 2, 3, 4, 25

Kingsum Chow and David Notkin. Semi-automatic update of applications in response to library changes. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 359–368, 1996. doi: 10.1109/ICSM.1996.565039. 2, 45, 46, 47, 55, 63, 93, 113, 183

J. Craig Cleaveland and Chandra M. R. Kintala. Tools for building application generators. *AT&T Technical Journal*, 67(4):46–58, 1988. 25

William W. Cohen. Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning*, pages 115–123, 1995. 172

Brad Cossette and Robert J. Walker. Polylingual dependency analysis using island grammars: A cost versus accuracy evaluation. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 214–223, 2007. doi: 10.1109/ICSM.2007.4362634. 3, 4, 25, 28, 31, 37, 41, 59, 67, 69, 71

Brad Cossette and Robert J. Walker. DSketch: Lightweight, adaptable dependency analysis. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 297–306, 2010. doi: 10.1145/1882291.1882335. 68, 212

Bradley Cossette, Robert Walker, and Rylan Cottrell. Using structural generalization to discover

replacement functionality for API evolution. Technical Report 2014-745-10, Department of Computer Science, University of Calgary, Calgary, Canada, May 2014. 174

Bradley E. Cossette and Robert J. Walker. Seeking the ground truth: A retroactive study on the evolution and migration of software libraries. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 55:1–55:11, 2012. doi: 10.1145/2393596.2393661. 15, 94, 123, 163, 167, 212

Rylan Cottrell, Joseph J. C. Chang, Robert J. Walker, and Jörg Denzinger. Determining detailed structural correspondence for generalization tasks. In *Proceedings of the European Software Engineering Conference/ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 165–174, 2007. doi: 10.1145/1287624.1287649. 131

Rylan Cottrell, Robert J. Walker, and Jörg Denzinger. Semi-automating small-scale source code reuse via structural correspondence. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 214–225, 2008. doi: 10.1145/1453101.1453130. viii, 7, 130, 131, 133, 156, 185

Barthélémy Dagenais and Laurie Hendren. Enabling static analysis for partial Java programs. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 313–328, 2008. 54

Barthélémy Dagenais and Martin Robillard. Recommending adaptive changes for framework evolution. *ACM Transactions on Software Engineering and Methodology*, 20(4):19:1–19:35, September 2011. doi: 10.1145/2000799.2000805. 2, 4, 5, 45, 47, 52, 53, 54, 64, 93, 99, 105, 137, 153, 155, 166, 170, 230

Premkumar T. Devanbu. GENOA: A customizable, front-end-retargetable source code analysis framework. *ACM Transactions on Software Engineering and Methodology*, 8(2):177–212, 1999. doi: 10.1145/304399.304402. 25, 27, 28, 30, 31, 35

Danny Dig and Ralph Johnson. How do APIs evolve? A story of refactoring. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(2):83–107, March 2006a. doi: 10.1002/smr.v18:2. 5, 15, 43, 50, 115, 124, 126, 213, 221

Danny Dig and Ralph Johnson. How do APIs evolve? A story of refactoring. *Journal of Software Maintenance: Research and Practice*, 18(2):83–107, March/April 2006b. doi: 10.1002/smr.v18:2. 59, 60, 61, 93, 95, 96, 98, 107, 119, 120, 213

Danny Dig, Can Comertoglu, Darko Marinov, and Ralph Johnson. Automated detection of refactorings in evolving components. In *Proceedings of the European Conference on Object-Oriented Programming*, volume 4067 of *Lecture Notes in Computer Science*, pages 404–428, 2006. ix, x, 43, 50, 51, 52, 60, 62, 93, 153, 172, 176, 213, 214, 216, 217, 218, 219, 220, 221, 223, 226, 284

Danny Dig, Stas Negara, Vibhu Mohindra, and Ralph Johnson. ReBA: Refactoring-aware binary adaptation of evolving libraries. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 441–450, 2008. doi: 10.1145/1368088.1368148. 46, 60, 63, 93

Beat Fluri, Michael Wursch, Martin Pinzger, and Harald C. Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Transactions on Software Engineering*, 33(11):725–743, November 2007. doi: 10.1109/TSE.2007.70731. 132

Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999. 110, 114, 282

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. doi: 10.1006/jcss.1997.1504. 171

Zachary P. Fry and Wesley Weimer. A human study of fault localization accuracy. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 1–10, 2010. doi: 10.1109/ICSM.2010.5609691. 2

Michael Furr and Jeffrey S. Foster. Polymorphic type inference for the JNI. In *Proceedings of the European Symposium on Programming*, pages 309–324, 2006. 4, 31, 37, 38, 67

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. 54

Emanuel Giger, Martin Pinzger, and Harald Gall. Predicting the fix time of bugs. In *Proceedings of the International Workshop on Recommendation Systems for Softare Engineering*, pages 52–56, 2010. doi: 10.1145/1808920.1808933. 43

Kirk Glerum, Kinshuman Kinshumann, Steve Greenberg, Gabriel Aul, Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt. Debugging in the (very) large: Ten years of implementation and experience. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, pages 103–116, 2009. doi: 10.1145/1629575.1629586. 43

Michael W. Godfrey and Lijie Zou. Using origin analysis to detect merging and splitting of source code entities. *IEEE Transactions on Software Engineering*, 31(2):166–181, February 2005. doi: 10.1109/TSE.2005.28. 5, 47, 48, 49, 52, 93, 132, 162

James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. *The Java Language Specification*. Addison-Wesley, Java SE 7 edition, 2012. URL http://docs.oracle.com/javase/specs/jls/se7/html/index.html. 15, 94, 96, 163

Judith E. Grass and Yih-Farn Chen. The C++ information abstractor. In *Proceedings of the USENIX C++ Technical Conference*, pages 265–278, 1990. 4, 25, 26, 28, 31, 32, 35

Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. Characterizing and predicting which bugs get fixed: An empirical study of Microsoft Windows. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, volume 1, pages 495–504, 2010. doi: 10.1145/1806799.1806871. 2

William G. J. Halfond and Alessandro Orso. AMNESIA: Analysis and monitoring for neutralizing SQL-injection attacks. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pages 174–183, 2005. doi: 10.1145/1101908.1101935. 38, 71

William G. J. Halfond, Alessandro Orso, and Panagiotis Manolios. Using positive tainting and syntax-aware evaluation to counter SQL injection attacks. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 175–185, 2006. 38

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009. 172

Robert W. Hasker. *The Replay of Program Derivations*. PhD thesis, University of Illinois at Urbana–Champaign, 1995. 130

Johannes Henkel and Amer Diwan. CatchUp!: Capturing and replaying refactorings to support API evolution. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 274–283, 2005. doi: 10.1145/1062455.1062512. 46, 60, 62, 63, 93, 183

Reid Holmes. *Pragmatic Software Reuse*. PhD thesis, University of Calgary, Calgary, Canada, November 2008. 44

Reid Holmes and Robert J. Walker. Supporting the investigation and planning of pragmatic reuse tasks. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 447–457, 2007. doi: 10.1109/ICSE.2007.83. 1, 2, 156, 183

Reid Holmes and Robert J. Walker. Systematizing pragmatic software reuse. *ACM Transactions on Software Engineering and Methodology*, 21(4):20:1–20:44, November 2012. doi: 10.1145/2377656.2377657. 55

Gérard Huet. *Résolution d'équations dans des langages d'ordre 1,2,...,ω*. Thèse de doctorat ès sciences mathématiques, Université Paris VII, September 1976. 130

Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, 1999. Section 2.7: The Evils of Duplication. 67

Daniel Jacobson, Greg Brail, and Dan Woods. *APIs: A Strategy Guide*. O'Reilly Media, Inc., 2011. ISBN 1449308929, 9781449308926. 44, 45

Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, September 2010. 171, 172, 178

George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995. 171

Ralph Johnson. Frameworks = (components + patterns). *Communications of the ACM*, 40(10): 39–42, October 1997. doi: 10.1145/262793.262799. 44, 61

Jan Jonas. Reverse engineering of framework API changes via mining instantiation code. Diploma thesis, Technische Universität Darmstadt, Darmstadt, Germany, September 2007. 123, 153, 176, 227, 228, 230, 237

Puneet Kapur. Refactoring references for library migration. Msc thesis, University of Calgary, Calgary, Canada, May 2010. 96

Puneet Kapur, Brad Cossette, and Robert J. Walker. Refactoring references for library migration. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 726–738, 2010. doi: 10.1145/1869459.1869518. 2, 5, 7, 43, 47, 54, 55, 56, 60, 61, 62, 64, 93, 95, 221

Miryung Kim, David Notkin, and Dan Grossman. Automatic inference of structural changes for matching across program versions. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 333–343, 2007. doi: 10.1109/ICSE.2007.20. 5, 49, 50, 52, 53, 54, 63, 93, 108, 165, 229

Sunghun Kim, Kai Pan, and E. James Whitehead, Jr. When functions change their names: Automatic detection of origin relationships. In *Proceedings of the Working Conference on Reverse Engineering*, pages 143–152, 2005. 5, 48, 49, 52, 62

Sunghun Kim, Thomas Zimmermann, Kai Pan, and E. James Whitehead, Jr. Automatic identification of bug-introducing changes. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pages 81–90, 2006. doi: 10.1109/ASE.2006.23. 43

Paul Klint, Ralf Lämmel, and Chris Verhoef. Toward an engineering discipline for grammarware. *ACM Transactions on Software Engineering and Methodology*, 14(3):331–380, July 2005. doi: 10.1145/1072997.1073000. 41, 69

Ron Kohavi. The power of decision tables. In *Proceedings of the European Conference on Machine Learning*, pages 174–189, 1995. 171

Kostas Kontogiannis, Panos Linos, and Kenny Wong. Comprehension and maintenance of large-scale multi-language software applications. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 497–500, 2006. 31, 67

Rainer Koppler. A systematic approach to fuzzy parsing. *Software: Practice & Experience*, 27(6): 637–649, June 1997. doi: 10.1002/(SICI)1097-024X(199706)27:6⟨637::AID-SPE99⟩3.0.CO;2-3. 4, 40, 69

Jeff Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, April 2007. doi: 10.1145/1232743.1232745. 44

Charles W. Krueger. Software reuse. *ACM Computing Surveys*, 24(2):131–183, June 1992. doi: 10.1145/130844.130856. 1, 20

William Landi. Undecidability of static analysis. *ACM Letters in Programming Languages and Systems*, 1(4):323–337, December 1992. doi: Undecidabilityofstaticanalysis. 3, 22, 23, 24

C. Larman. Protected variation: The importance of being closed. *IEEE Software*, 18(3):89–91, May 2001. doi: 10.1109/52.922731. 93

M. M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, September 1980. doi: 10.1109/PROC.1980.11805. 1, 93

M. M. Lehman. Laws of software evolution revisited. In *Proceedings of the European Workshop on Software Process Technology*, volume 1149 of *Lecture Notes in Computer Science*, 1996. 1

Vladimir I. Levenshtein. Binary ccode capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966. 164

Sheng Liang. *Java Native Interface: Programmer's Guide and Reference*. Addison-Wesley, 1999. ISBN 0201325772. 32

Luqi. A graph model for software evolution. *IEEE Transactions on Software Engineering*, 16(8): 917–927, August 1990. doi: 10.1109/32.57627. 2, 22, 25

Evan Martin and Tao Xie. Understanding software application interfaces via string analysis. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 901–904, 2006. doi: 10.1145/1134285.1134447. 38, 39

M. D. McIlroy. Mass-produced software components. In *Software Engineering: Report on a Conference by the NATO Science Committee*, pages 138–155, 1968. URL http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF. 44

Sichen Meng, Xiaoyin Wang, Lu Zheng, and Hong Mei. A history-based matching approach to identification of framework evolution. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 353–363, 2012. doi: 10.1109/ICSE.2012.6227179. ix, x, xi, 54, 55, 63, 123, 124, 125, 153, 176, 213, 214, 266, 267, 271, 279, 280, 281, 282, 283, 284, 288

D. L. Moise and K. Wong. Extracting and representing cross-language dependencies in diverse software systems. In *Proceedings of the Working Conference on Reverse Engineering*, pages 209–218, 2005. 4, 31, 35, 36, 37, 67

Leon Moonen. Generating robust parsers using island grammars. In *Proceedings of the Working Conference on Reverse Engineering*, pages 13–24, 2001. 27, 40, 41, 69, 71

Leon Moonen. Lightweight impact analysis using island grammars. In *Proceedings of the IEEE International Workshop on Program Comprehenension*, pages 219–228, 2002. 4, 27, 40, 41, 67, 69, 71

Brandon Morel and Perry Alexander. SPARTACAS: Automating component reuse and adaptation. *IEEE Transactions on Software Engineering*, 30(9):587–600, 2004. doi: 10.1109/TSE.2004.53. 93

M. Moriconi and T. C. Winkler. Approximate reasoning about the semantic effects of program changes. *IEEE Transactions on Software Engineering*, 16(9):980–992, 1990. 2, 3, 20, 22, 23, 24, 25

Gail C. Murphy and David Notkin. Lightweight lexical source model extraction. *ACM Transactions on Software Engineering and Methodology*, 5(3):262–292, 1996. 4, 29, 33, 34, 35, 67

Gail C. Murphy, Robert J. Walker, and Elisa L. A. Baniassad. Evaluating emerging software development technologies: Lessons learned from assessing aspect-oriented programming. *IEEE Transactions on Software Engineering*, 25(4):438–455, July/August 1999. doi: 10.1109/32. 799936. Special Section on Empirical Software Engineering. 90

Emerson R. Murphy-Hill, Thomas Zimmermann, Christian Bird, and Nachiappan Nagappan. The design of bug fixes. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 332–341, 2013. 2

Kıvanç Muşlu, Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Speculative analysis of integrated development environment recommendations. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2012. doi: 10.1145/2384616.2384665. 185

Nachiappan Nagappan, Andreas Zeller, Thomas Zimmermann, Kim Herzig, and Brendan Murphy. Change bursts as defect predictors. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 309–318, 2010. doi: 10.1109/ISSRE.2010.25. 43

Hoan Anh Nguyen, Tung Thanh Nguyen, Gary Wilson, Jr., Anh Tuan Nguyen, Miryung Kim, and Tien N. Nguyen. A graph-based approach to API usage adaptation. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 302–321, 2010. doi: 10.1145/1869459.1869486. 53, 62, 63

Marius Nita and David Notkin. Using twinning to adapt programs to alternative APIs. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 205–214, 2010. doi: 10.1145/1806799.1806832. 5, 47, 55, 56, 64, 93, 113, 185

Cosmin Oancea, Clare So, and Stephen M. Watt. Generalization in Maple. In *Proceedings of the Maple Conference*, pages 277–382, 2005. URL http://www.csd.uwo.ca/~watt/pub/reprints/2005-mc-gen.pdf. 131

William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1992. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.914&rep=rep1&type=pdf. 49

Cherri M. Pancake. The promise and the cost of object technology: A five-year forecast. *Communications of the ACM*, 38(10):32–49, October 1995. doi: 10.1145/226239.226247. 4, 44, 45

Jihun Park, Miryung Kim, Baishakhi Ray, and Doo-Hwan Bae. An empirical study of supplementary bug fixes. In *Proceedings of the International Working Conference on Mining Software Repositories*, pages 40–49, 2012. doi: 10.1109/MSR.2012.6224298. 2

D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972. doi: 10.1145/361598.361623. 1, 3, 21

D. L. Parnas and D. P. Siewiorek. Use of the concept of transparency in the design of hierarchically structured systems. *Communications of the ACM*, 18(7):401–408, July 1975. doi: 10.1145/360881.360913. 1, 21, 44

David L. Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, 5(2):128–138, 1979. 1, 3, 21

David Lorge Parnas. Software aging. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 279–287, 1994. doi: 10.1109/ICSE.1994.296790. 1, 2, 3, 21, 43, 46, 93

Jeff Perkins. Automatically generating refactorings to support API evolution. In *Proceedings of the ACM SIGPLAN–SIGSOFT Workshop on Program Analysis and Software Tools Engineering*, pages 111–114, 2005. doi: 10.1145/1108792.1108818. 4, 45, 47, 55, 62, 93, 112

Jeff H. Perkins, Sunghun Kim, Sam Larsen, Saman Amarasinghe, Jonathan Bachrach, Michael Carbin, Carlos Pacheco, Frank Sherwood, Stelios Sidiroglou, Greg Sullivan, Weng-Fai Wong, Yoav Zibin, Michael D. Ernst, and Martin Rinard. Automatically patching errors in deployed software. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, pages 87–102, 2009. doi: 10.1145/1629575.1629585. 43

Paul J. Perrone, Venkata S. R. R. Chaganti, and Tom Schwenk. *J2EE Developer's Handbook*. Sams Publishing, 2003. 31

Gordon D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970. 130

Andy Podgurski and Lori A. Clarke. A formal model of program dependences and its implications for software testing, debugging, and maintenance. *IEEE Transactions on Software Engineering*, 16(9):965–979, 1990. 2, 3, 22, 23, 24, 25

Jeffrey S. Poulin, Joseph M. Caruso, and Debera R. Hancock. The business case for software reuse. *IBM Systems Journal*, 32(4):567–594, 1993. doi: 10.1147/sj.324.0567. 44

Thomas Reps and Tim Teitelbaum. The synthesizer generator. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 42–48, 1984. doi: 10.1145/390010.808247. 25

Howard B. Reubenstein, Richard L. Piazza, and Susan N. Roberts. Separating parsing and analysis in reverse engineering tools. In *Proceedings of the Working Conference on Reverse Engineering*, pages 117–125, 1993. 4, 25, 26, 27, 28, 31, 32, 35, 67

M. Robillard, E. Bodden, D. Kawrykow, M. Mezini, and T. Ratchford. Automated API property inference techniques. *IEEE Transactions on Software Engineering*, 39(5):613–637, 2013. 15

Martin Robillard and Robert DeLine. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011. 100, 105

Martin P. Robillard and Gail C. Murphy. Representing concerns in source code. *ACM Transactions on Software Engineering and Methodology*, 16(1):3:1–3:38, 2007. 2

Thorsten Schäfer, Jan Jonas, and Mira Mezini. Mining framework usage changes from instantiation code. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 471–480, 2008. doi: 10.1145/1368088.1368153. ix, x, 5, 53, 54, 63, 95, 99, 119, 121, 122, 123, 126, 137, 151, 153, 154, 160, 164, 166, 170, 172, 176, 177, 213, 214, 227, 228, 229, 230, 231, 235, 236, 237, 266, 284, 289

W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 13(2): 115–139, 1974. doi: 10.1147/sj.132.0115. 3, 21

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society Series B (Methodology)*, 36(2):111–147, 1974. URL http://www.jstor.org/stable/2984809. 173

Anselm L. Strauss. *Continual Permutations of Action*. Aldine de Gruyter, New York, 1993. ISBN 0202304728. 1

Bjarne Stroustrup. Evolving a language in and for the real world: C++ 1991–2006. In *Proceedings of the ACM SIGPLAN Conference on History of Programming Languages*, pages 1–59, 2007. doi: 10.1145/1238844.1238848. 26, 27

Nikita Synytskyy, James R. Cordy, and Thomas R. Dean. Robust multilingual parsing using island grammars. In *Proceedings of the IBM Centre for Advanced Studies Conference on Collaborative Research*, pages 266–278, 2003. 41

Craig Taube-Schock, Robert J. Walker, and Ian H. Witten. Can we avoid high coupling? In *Proceedings of the European Conference on Object-Oriented Programming*, volume 6813 of *Lecture Notes in Computer Science*, pages 204–228, 2011. doi: 10.1007/978-3-642-22655-7_10. 1

J. van Gurp and J. Bosch. Design, implementation and evolution of object oriented frameworks: Concepts and guidelines. *Software: Practice & Experience*, 31(3):277–300, March 2001. doi: 10.1002/spe.366. 2, 44

Mathieu Verbaere, Ran Ettinger, and Oege de Moor. JunGL: A scripting language for refactoring. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 172–181, 2006. 98

Ulrich Wagner. Combinatorically restricted higher order anti-unification: An application to programming by analogy. Master's thesis, Technische Universität Berlin, Berlin, Germany, April 2002. URL http://www2.informatik.uni-osnabrueck.de/schmid/tubhp/diparbstub/wagner/DIPLOM.pdf. 130

Larry Wall and Randal L Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., 1991. 184

Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984. 54

Peter Weißgerber and Stephan Diehl. Identifying refactorings from source-code changes. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pages 231–240, 2006. doi: 10.1109/ASE.2006.41. 49, 50, 51, 52, 54, 60, 62, 63, 96, 98

Niklaus Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14 (4):221–227, 1971. doi: 10.1145/362575.362577. 21

Kenny Wong, Scott R. Tilley, Hausi A. Müller, and Margaret-Anne D. Storey. Structural redocumentation: A case study. *IEEE Software*, 12(1):46–54, 1995. doi: 10.1109/52.363166. 4, 28, 31, 32, 39, 67, 69

Wei Wu, Yann-Gaël Guéhéneuc, Giuliano Antoniol, and Miryung Kim. AURA: A hybrid approach to identify framework evolution. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, volume 1, pages 325–334, 2010. doi: 10.1145/1806799.1806848. ix, x, 5, 47, 53, 54, 63, 93, 95, 99, 119, 123, 124, 125, 126, 127, 151, 152, 153, 156, 161, 164, 166, 170, 172, 176, 213, 214, 237, 238, 241, 242, 243, 244, 250, 251, 262, 265, 266, 284

Zhenchang Xing and Eleni Stroulia. UMLDiff: An algorithm for object-oriented design differencing. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pages 54–65, 2005. 48, 52, 63

Zhenchang Xing and Eleni Stroulia. Differencing logical UML models. *Automated Software Engineering: An International Journal*, 14(2):215–259, 2007a. 51

Zhengchang Xing and Eleni Stroulia. API-evolution support with Diff-CatchUp. *IEEE Transactions on Software Engineering*, 33(12):818–836, 2007b. doi: 10.1109/TSE.2007.70747. 2, 4, 5, 15, 45, 47, 51, 52, 53, 63, 93, 166

Zuoning Yin, Ding Yuan, Yuanyuan Zhou, Shankar Pasupathy, and Lakshmi Bairavasundaram. How do fixes become bugs? In *Proceedings of the European Software Engineering Conference/ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 26–36, 2011. doi: 10.1145/2025113.2025121. 43

Hongyu Zhang, Liang Gong, and Steve Versteeg. Predicting bug-fixing time: An empirical study of commercial software projects. In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, pages 1042–1051, 2013. URL http://dl.acm.org/citation.cfm?id=2486788.2486931. 43

George Kingsley Zipf. *Human behavior and the principle of least effort.* Addison-Wesley Press, 1949. 214

Appendix A

Overview of Appendices

For the sake of the reader, I provide an overview of the appendices here.

In Appendix B, I list the file number and application title of the ethics review received to conduct my case study in Chapter 4. Appendix C provides the copyright details of my published papers that form a part of this dissertation; the copyright notices also appear as footnotes in the relevant chapters as requested by the ACM Copyright Policy.[1]

A detailed comparison of the changes and replacements recorded in the Library Migration Corpus, including comparative examples showing where the results of the Corpus disagree with the existing literature, is provided in Appendix D. The entirety of version 2.0 of the Library Migration Corpus is included as Appendix E.

[1] http://www.acm.org/publications/policies/copyright_policy, version 8 was followed.

Appendix B

Certification of Ethics Approval

The evaluation of the DSKETCH tool in Chapter 4 was approved by the Conjoint Faculties Research Ethics Board for the University of Calgary, under File no: 6926, *"Qualitative Studies into Software Development Practices"*. A copy of the certification of institutional ethics review was submitted along with this dissertation.

Appendix C

List of Publications and Copyright Permissions

In this section, I list those papers which I have authored or co-authored, whose material forms a part of this thesis. The appropriate copyright permissions are also included.

DSKETCH: Lightweight, adaptable dependency analysis. Cossette and Walker [2010] ©2010 ACM. This is a minor revision of the work published in the *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2010, http://doi.acm.org/10.1145/1882291.1882335. Included here by permission. Used in Chapter 4.

Seeking the ground truth: A retroactive study on the evolution and migration of software libraries. Cossette and Walker [2012] ©2012 ACM. This is a minor revision of the work published in the *Proceedings of the Twentieth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2012, http://doi.acm.org/10.1145/2393596.2393661. This work was awarded an ACM Distinguished Paper Award, at the ACM SIGSOFT International Symposium on Foundations of Software Engineering, Carey, North Carolina, November 2012. Included here by permission. Used in Chapter 5.

Appendix D

Comparing Existing Work to the Library Migration Corpus

In Chapters 5, 6, and 7, I compare the Library Migration Corpus ("the Corpus"). I report the effectiveness of several tools and techniques as being significantly different than their authors did in their publications. In this appendix, I compare in detail the reported changes from the Library Migration Corpus to those found by Dig et al. [2006], Schäfer et al. [2008], and Wu et al. [2010].

The rest of this appendix is structured as follows. In Section D.1 I provide an overview and introduction to the appendix, and why these particular results are being compared. I continue by comparing the Corpus to the results of existing approaches as follows: Dig et al. in Section D.2; Schäfer et al. in Section D.3; and Wu et al. in Section D.4. Because this appendix is unavoidably lengthy, I provide a summary at the end of each section of the overall comparison.

D.1 Overview

The work of Dig and Johnson [2006b] has proven to be influential to later works in origin analysis and API change recommenders in many aspects; one was their usage of the APACHE STRUTS library migration from version 1.1.0 to 1.2.4 in their empirical study [Dig and Johnson, 2006a], and subsequent evaluation of the REFACTORINGCRAWLER tool [Dig et al., 2006]. This same library migration was later used in the evaluations of Schäfer et al. [2008], Wu et al. [2010], and Meng et al. [2012]: the four other libraries that Dig and Johnson examined have been used far less frequently. While STRUTS is an older framework[1], including it in the corpus gave me the opportunity to replicate the work of other researchers in this area, and to contrast their results with my own.

While the Corpus restricts its consideration of library changes to binary incompatibilities be-

[1] Initially released in 2000, the final release of the STRUTS 1 framework was in 2008. http://struts.apache.org/struts1eol-announcement.html

tween API versions, the evaluations of the above techniques did not: additional non-BI library changes are included in their results, comprising four categories.

1. API changes that did not result in a binary incompatibility, and thus do not break backwards compatibility [Dig et al., 2006; Schäfer et al., 2008; Wu et al., 2010; Meng et al., 2012].

2. Changes in how the API is used, independent of an actual API change [Dig et al., 2006; Schäfer et al., 2008; Wu et al., 2010].

3. Changes to the library's implementation that are concealed by its API, and for which the API does not change [Schäfer et al., 2008; Wu et al., 2010; Meng et al., 2012].

4. Changes to test cases, example applications, or functionality from other related libraries [Wu et al., 2010; Meng et al., 2012].

My choice to restrict the Corpus to binary incompatible changes between library versions stems from its usefulness in mechanically defining the minimal set of changes that are *necessary* but not *sufficient* to redress for library migration (i.e., these changes at minimum must be redressed for a successful migration, but there may exist other changes that altered the API's behaviour). Non-API changes are not considered for inclusion in the Corpus: this follows from the reasonable assumption that developers will apply the principle of least effort [Zipf, 1949] in migrating their source code, and will focus on repairing broken functionality, rather than altering functional code that now has alternatives.

In comparing the reported results of other tools to the Corpus, a dramatic reduction in the accuracy of recommended changes for the STRUTS 1.1.0 to 1.2.4 migration is observed. In part, this reflects the restriction of considered changes to those resulting in a BI between versions, but it is reasonable to wonder if the changes detailed in the Corpus are wrong, incomplete, or present only one of several possible correct adaptations.

Ideally, a series of unit tests would exist for every BI API change that showed: (i) the correct functioning of a test prior to the migration; (ii) its failure post-migration; and, (iii) its successful redress using the change detailed in the Corpus. Since few of the libraries considered contained an associated test suite, and those test suites which were provided covered a small fraction of the affected API, the costs of providing such detailed evidence would be expensive.

Alternatively, I provide in this chapter a detailed comparison of the differences between changes reported by the Corpus and those of the examined techniques, with an in depth examination of representative changes, to show that:

1. the reported results of the Corpus, when compared to those of the above techniques, are correct;

2. discrepancies in the findings of other approaches are due to either a mistake in their analysis, their consideration of non-BI changes, or both; and,

3. the restriction to consider only BI changes provides an impartial standard for evaluating technique accuracy.

D.1.1 On Notation

In referring to methods in this writeup, I use the notation "Classname.methodname()" or "methodname ()"; in so doing, I omit the possible list of parameters the method may accept. This is done for the convenience of the reader and the writer, as the presentation would be unnecessarily cluttered otherwise. Similarly, the fully qualified names of Classes have been truncated in some tables to facilitate the presentation of these results: a notation that all listed Classes are in a specific parent package accompanies such tables.

D.2 Comparison with Dig et al.

Dig et al. have published the detailed predictions of their REFACTORINGCRAWLER tool for the STRUTS 1.1.0 to 1.2.4 migration online;[2] the comparisons in this section are taken from this dataset. REFACTORINGCRAWLER reports refactorings it detects between versions of the library's API; I compare these detected refactorings against those API changes recorded in the Corpus.

The data presented in Table D.1, Table D.2, and Table D.3 is presented as recorded in the provided data, with the exception that I omit their classification of the refactoring identified. Dig et al. report REFACTORINGCRAWLER as having 100% precision, and 86% recall when identifying API refactorings in the STRUTS 1.1.0 to 1.2.4 library migration.

D.2.1 Changes on which the Corpus and Dig et al. Agree

Table D.1 lists the API changes detected by REFACTORINGCRAWLER that the Corpus also identifies as BI changes, and for which the recommended replacement is in agreement. It should be noted that when REFACTORINGCRAWLER reported a replacement for a binary incompatible to the STRUTS API, it agreed with the Corpus in every case.

Table D.1: Dig et al. Matched API Changes. (All in org.apache.struts)

Original	Replacement
taglib.html.FrameTag.setMarginheight	taglib.html.FrameTag.setMarginheight
taglib.html.FrameTag.setMarginwidth	taglib.html.FrameTag.setMarginwidth
tiles.UntyppedAttribute	tiles.UntypedAttribute
validator.FieldChecks.validateByte	validator.FieldChecks.validateByte
validator.FieldChecks.validateCreditCard	validator.FieldChecks.validateCreditCard
validator.FieldChecks.validateDate	validator.FieldChecks.validateDate
validator.FieldChecks.validateDouble	validator.FieldChecks.validateDouble

Continued on next page

[2] http://dig.cs.illinois.edu/tools/RefactoringCrawler/evaluations.html

Table D.1 — continued from previous page

Original	Replacement
validator.FieldChecks.validateDoubleRange	validator.FieldChecks.validateDoubleRange
validator.FieldChecks.validateEmail	validator.FieldChecks.validateEmail
validator.FieldChecks.validateFloat	validator.FieldChecks.validateFloat
validator.FieldChecks.validateFloatRange	validator.FieldChecks.validateFloatRange
validator.FieldChecks.validateInteger	validator.FieldChecks.validateInteger
validator.FieldChecks.validateIntRange	validator.FieldChecks.validateIntRange
validator.FieldChecks.validateLong	validator.FieldChecks.validateLong
validator.FieldChecks.validateMask	validator.FieldChecks.validateMask
validator.FieldChecks.validateMaxLength	validator.FieldChecks.validateMaxLength
validator.FieldChecks.validateMinLength	validator.FieldChecks.validateMinLength
validator.FieldChecks.validateRequired	validator.FieldChecks.validateRequired
validator.FieldChecks.validateShort	validator.FieldChecks.validateShort
validator.Resources.initValidator	validator.Resources.initValidator

D.2.2 Non-Library Changes Considered by Dig et al.

Table D.2 lists changes that REFACTORINGCRAWLER detects that are not to the STRUTS library API per se, but rather to the STRUTS-TILES framework[3] that was a companion to STRUTS v1.1.0. Because this was not part of the core STRUTS library, I excluded it from analysis, and cannot speak to whether these represent binary incompatibilities in that framework. The Corpus only considered changes to the STRUTS API itself.

[3] http://tiles.apache.org/faq.html

Table D.2: Dig et al. Non-Library Changes. (All in org.apache.struts.webapp.tiles)

Original	Replacement
channel.SelectChannelAction.perform	channel.SelectChannelAction.execute
dynPortal.RetrievePortalAction.perform	dynPortal.RetrievePortalAction.execute
dynPortal.SetPortalPrefsAction.perform	dynPortal.SetPortalPrefsAction.execute
invoice.EditInvoiceAction.perform	invoice.EditInvoiceAction.execute
lang.SelectLocaleAction.perform	lang.SelectLocaleAction.execute
portal.UserMenuAction.perform	portal.UserMenuAction.execute
portal.UserMenuSettingsAction.perform	portal.UserMenuSettingsAction.execute
portal.UserPortalAction.perform	portal.UserPortalAction.execute
portal.UserPortalSettingsAction.perform	portal.UserPortalSettingsAction.execute
rssChannel.Channels.perform	rssChannel.Channels.execute
rssChannel.RssChannelsAction.perform	rssChannel.RssChannelsAction.execute
skin.LayoutSettingsAction.perform	skin.LayoutSettingsAction.execute
skin.LayoutSwitchAction.perform	skin.LayoutSwitchAction.execute
skin.SimpleSwitchLayoutAction.perform	skin.SimpleSwitchLayoutAction.execute
template.DynTemplateAction.perform	template.DynTemplateAction.execute
test.TestActionTileAction.perform	test.TestActionTileAction.execute
tutorial.ForwardExampleAction.perform	tutorial.ForwardExampleAction.execute

D.2.3 Non-BI Changes Considered by Dig et al.

Table D.3 lists changes which REFACTORINGCRAWLER detects in the Struts API that are not reported by the Corpus. There are three causes for these differences.

Table D.3: Dig et al. Non-BI Changes. (All in org.apache.struts)

Original	Replacement
Deprecated in 1.2.4, present and undeprecated in 1.3.10.	
action.Action.saveErrors	action.Action.saveErrors
Not deprecated, never removed, present in 1.3.10.	
util.RequestUtils.forwardURL	util.RequestUtils.forwardURL
Pull-up method refactoring, present in superclass in 1.3.10	
action.ActionMapping.findException	config.ActionConfig.findException
Mistake: identified a new method addition as a method rename.	
tiles.ActionController.perform	tiles.ActionController.execute
tiles.Controller.perform	tiles.Controller.execute
tiles.UrlController.perform	tiles.UrlController.execute

Continued on next page

Table D.3 — continued from previous page

Original	Replacement
Deprecated, but not removed until sometime between the 1.2.7 and 1.3.0 release.	
action.ActionForward.ActionForward	action.ActionForward.ActionForward
action.ExceptionHandler.storeException	action.ExceptionHandler.storeException
config.ForwardConfig.ForwardConfig	config.ForwardConfig.ForwardConfig
util.RequestUtils.encodeURL	taglib.TagUtils.encodeURL
util.RequestUtils.getActionMappingName	taglib.TagUtils.getActionMappingName
util.RequestUtils.getActionMessages	taglib.TagUtils.getActionMessages
util.RequestUtils.getModuleConfig	taglib.TagUtils.getModuleConfig
util.RequestUtils.getModuleConfig	taglib.TagUtils.getModuleConfig
util.RequestUtils.getModuleName	taglib.TagUtils.getModuleName
util.RequestUtils.getModuleName	taglib.TagUtils.getModuleName
util.RequestUtils.getScope	taglib.TagUtils.getScope
util.RequestUtils.isXhtml	taglib.TagUtils.isXhtml
util.RequestUtils.lookup	taglib.TagUtils.lookup
util.RequestUtils.lookup	taglib.TagUtils.lookup
util.RequestUtils.message	taglib.TagUtils.message
util.RequestUtils.message	taglib.TagUtils.message
util.RequestUtils.present	taglib.TagUtils.present
util.RequestUtils.saveException	taglib.TagUtils.saveException
util.RequestUtils.selectModule	taglib.TagUtils.selectModule
util.RequestUtils.selectModule	taglib.TagUtils.selectModule
util.ResponseUtils.write	taglib.TagUtils.write
util.ResponseUtils.writePrevious	taglib.TagUtils.writePrevious

1. REFACTORINGCRAWLER identifies refactorings in cases where code is not removed. Dig et al. report cases where they detect deprecated functionality and a provided replacement, but for which the deprecated functionality was not removed, as a refactoring on the API. They assume that these cases represent a stage in the "Deprecate-Replace-Remove" cycle [Dig and Johnson, 2006a]: the entity's replacement has been provided, and it will be removed in a subsequent version. For most of these changes, this is the case: while the Corpus never reports these entities as causing a binary incompatibility in the versions of STRUTS examined,[4] they are eventually removed prior to the final release of STRUTS 1. Until then, the original functionality remains and continues to provide the same behaviour as before. The Corpus does not report these as BI API changes in the versions examined: while the affected code will eventually be removed, they are not *necessary* to enact for migrating between any of the versions of STRUTS the Corpus considers. If the Corpus is extended to cover releases in the STRUTS library after version 1.3, it would then identify these as BI changes in version from which they are removed.

There are cases in which REFACTORINGCRAWLER's identification of API refactorings proves to be premature. One is the org.apache.struts.action.Action.saveErrors(HttpServletRequest,ActionErrors) method from STRUTS 1.1.0, presented in Listing D.1; the method's implementation in the 1.2.4 version of STRUTS is presented in Listing D.2. Notable differences between the two are that the method has been marked deprecated, its functionality is delegated to another method, and it appears ready to be removed from the library. For this reason, Dig et al. mark this method as a refactoring in the 1.1.0 to 1.2.4 migration, though it has not yet necessitated an API change.

However, as Kapur et al. [2010] note, deprecated functionality can remain in the API for several versions, and may never be removed: it is not to see the "@deprecated" tag removed from such methods in later versions. That is the case here.

Despite the comment associated with the @deprecated tag in Listing D.2 that states the method will be removed "after STRUTS 1.2", the functionality *remained* in the library, and was present in

[4]The Corpus does not report these as binary incompatible API changes, as they are not removed in the versions examined.

```
/**
 * Save the specified error messages keys into the appropriate request
 * attribute for use by the &lt;html:errors&gt; tag, if any messages
 * are required. Otherwise, ensure that the request attribute is not
 * created.
 *
 * @param request The servlet request we are processing
 * @param errors Error messages object
 */
protected void saveErrors(HttpServletRequest request,
          ActionErrors errors) {

  // Remove any error messages attribute if none are required
  if ((errors == null) || errors.isEmpty()) {
    request.removeAttribute(Globals.ERROR_KEY);
    return;
  }

  // Save the error messages we need
  request.setAttribute(Globals.ERROR_KEY, errors);

}
```

Listing D.1: Action.saveErrors(), in STRUTS v1.1.0.

```
/**
 * <p>Save the specified error messages keys into the appropriate request
 * attribute for use by the &lt;html:errors&gt; tag, if any messages
 * are required. Otherwise, ensure that the request attribute is not
 * created.</p>
 *
 * @param request The servlet request we are processing
 * @param errors Error messages object
 * @deprecated Use saveErrors(HttpServletRequest, ActionMessages) instead.
 * This will be removed after Struts 1.2.
 */
protected void saveErrors(HttpServletRequest request, ActionErrors errors) {

  this.saveErrors(request,(ActionMessages)errors);
  // :TODO: Remove after Struts 1.2.

}
```

Listing D.2: Action.saveErrors(), in STRUTS v1.2.4.

the final release of the STRUTS 1 framework, v1.3.10.[5] Listing D.3 presents the 1.3.10 version of the saveErrors method. The @deprecated tag has been removed, and the final implementation is no

[5]Details on the end-of-life of STRUTS 1.x, including the source code of the last release (v1.3.10) can be found at http://struts.apache.org/download.cgi.

```
1  /**
2   * <p>Save the specified error messages keys into the appropriate request
3   * attribute for use by the &lt;html:errors&gt; tag, if any messages are
4   * required. Otherwise, ensure that the request attribute is not
5   * created.</p>
6   *
7   * @param request The servlet request we are processing
8   * @param errors Error messages object
9   * @since Struts 1.2
10  */
11 protected void saveErrors(HttpServletRequest request, ActionMessages errors) {
12    // Remove any error messages attribute if none are required
13    if ((errors == null) || errors.isEmpty()) {
14       request.removeAttribute(Globals.ERROR_KEY);
15
16       return;
17    }
18
19    // Save the error messages we need
20    request.setAttribute(Globals.ERROR_KEY, errors);
21 }
```

Listing D.3: Action.saveErrors(), in STRUTS v1.3.10.

longer delegated, but returned to its original implementation. Because Dig et al. only examined one migration, they could not have noticed this functionality was not removed in the next release.

In another case, Dig et al. recorded a delegated change that was not marked as deprecated as a refactoring: the org.apache.struts.util.RequestUtils.forwardURL() method. As illustrated by Listing D.4 and Listing D.5, the functionality in the method's 1.1.0 implementation is delegated in the next STRUTS release, which is why REFACTORINGCRAWLER identifies this as an API refactoring. However, this functionality is never removed, and is present in the 1.3.10 release. The Corpus does not report this as a BI API change: it is not *necessary* to migrate code dependent on this functionality to its designated replacement, as the existing functionality remains in the STRUTS API.

Attempts to report library changes based on the presence of @deprecated tags or replacement functionality, where no API change has yet occurred, must the non-negligible risk the proposed change will never occur.

```
1  public static String forwardURL(HttpServletRequest request, ForwardConfig forward) {
2
3    String path = forward.getPath();
4
5    // Handle a ForwardConfig marked as context relative
6    StringBuffer sb = new StringBuffer();
7    if (forward.getContextRelative()) {
8      if (!path.startsWith("/")) {
9        sb.append("/");
10     }
11     sb.append(path);
12     return (sb.toString());
13   }
14   ...
15   //Rest of implementation elided to simplify presentation
16 }
```

Listing D.4: RequestUtils.forwardURL(), v1.1.0

```
1 public static String forwardURL(HttpServletRequest request, ForwardConfig forward) {
2    return forwardURL(request,forward,null);
3 }
```

Listing D.5: RequestUtils.forwardURL(), v1.2.4

2. REFACTORINGCRAWLER identifies refactorings that are backwards compatible. The org.apache.struts.action.ActionMapping.findException() method, which REFACTORINGCRAWLER correctly reports as a *pull up method* refactoring into the org.apache.struts.config.ActionConfig class, is a compatible change to the API. The ActionConfig class is the *parent* of the ActionMapping class, so the movement of the findException() method into ActionConfig will cause ActionMapping to now inherit the method, thus making this a backwards-compatible change to the API. The Corpus does not report this as a BI API change, as code dependent on this method can continue to function without alteration.

3. Errors by REFACTORINGCRAWLER *in identifying changes.* There are three related cases where discrepancies between the Corpus and REFACTORINGCRAWLER indicates REFACTORINGCRAWLER made an error. These have to do with the org.apache.struts.tiles.Controller interface, and two classes that implement it. REFACTORINGCRAWLER records the Controller.perform() method as being renamed to Controller.execute(), and marks its implementing classes in the same way. How-

ever, this is not the case.

Listing D.6 shows the entirety of the Controller interface in STRUTS v1.1.0 (but elides the **package** and **import** statements), while Listing D.7 shows the revised interface in v1.2.4. While an execute() method has been added, and contains the same parameter types as the perform() method, this is not a rename refactoring: the exception lists for both methods are different.[6] Both methods in Controller are present in version 1.3.10 of STRUTS. I suggest this should be reported as a false-positive in the REFACTORINGCRAWLER's analysis.

The Corpus does not report an API change for Controller.perform(), or the two implementing subclasses; instead, it notes that the addition of the Controller.execute() method is a binary incompatible modification to the Controller interface. Any classes in the developer's source code that implemented the Controller class must implement this new method as well.

```
1  /**
2   * A controller is a piece of code called before rendering a jsp page.
3   * A controller can be associated to a tile. See <insert> or <definition> for
4   * association syntax.
5   */
6  public interface Controller
7  {
8
9      /**
10      * Method associated to a tile and called immediately before the tile is included.
11      * @param tileContext Current tile context.
12      * @param request Current request
13      * @param response Current response
14      * @param servletContext Current servlet context
15      */
16     public void perform(ComponentContext tileContext,
17                         HttpServletRequest request,
18                         HttpServletResponse response,
19                         ServletContext servletContext)
20         throws ServletException, IOException;
21 }
```

Listing D.6: Controller, v1.1.0

[6] This is a technical but non-trivial distinction. It is likely the execute() method was introduced to support a wider variety of possible exceptions, as it declares that it throws the Exception superclass, rather than the specific exception types from the perform() method. Simply renaming functionality to use this new method would be dangerous, and should introduce a compilation error, as developers need to ensure that additional exceptions are accounted for in their code.

```
/**
 * A controller is a piece of code called before rendering a jsp page.
 * A controller can be associated to a tile. See &lt;insert&gt; or
 * &lt;definition&gt; for association syntax.
 */
public interface Controller {

  /**
   * Method associated to a tile and called immediately before the tile
     * is included.
   * @param tileContext Current tile context.
   * @param request Current request
   * @param response Current response
   * @param servletContext Current servlet context
     * @deprecated Use execute() instead. This will be removed after
     * Struts 1.2.
   */
  public void perform(
    ComponentContext tileContext,
    HttpServletRequest request,
    HttpServletResponse response,
    ServletContext servletContext)
    throws ServletException, IOException;

    /**
     * Method associated to a tile and called immediately before the tile
     * is included.
     * @param tileContext Current tile context.
     * @param request Current request
     * @param response Current response
     * @param servletContext Current servlet context
     */
    public void execute(
        ComponentContext tileContext,
        HttpServletRequest request,
        HttpServletResponse response,
        ServletContext servletContext)
        throws Exception;
}
```

Listing D.7: Controller, v1.2.4

D.2.4 Comparison with Dig et al.: Summary

Table D.4: Comparison of REFACTORINGCRAWLER to the Corpus for STRUTS 1.1.0 to 1.2.4.

Changes Identified	BI Changes		Non-BI Changes			Non-STRUTS Changes
	Agrees	Disagrees	Later Removed	Never Removed	Error	
65	20	0	22	3	3	17

Table D.4 summarizes the comparison of the REFACTORINGCRAWLER tool's identified library changes, and replacements, to those of the Corpus for the STRUTS v1.1.0 to v1.2.4 migration. Of

the 65 changes REFACTORINGCRAWLER identifies, 20 are in agreement with the Corpus. Of the remainder, 17 involve changes to a library collocated with STRUTS, and 28 do not result in a binary incompatible change.

Of the refactorings that REFACTORINGCRAWLER identified in the STRUTS API, 28 did not actually result in a BI change to the API, as the original API methods were not removed in this migration. For 22 of these, the methods are removed some time prior to the final release of the STRUTS API, but not in the versions the Corpus examines. Three identified refactorings involve API functionality that is never removed, either because: the refactoring was backwards compatible; the developers reversed a decision to deprecate functionality; or, the functionality was never removed from the library. The final three refactorings identified are cases where I believe REFACTORINGCRAWLER erroneously classifies a new method addition as a rename refactoring.

In reporting on the results of REFACTORINGCRAWLER in Chapters 6 and 7, I have excluded the results listed in Section D.2.2 and Section D.2.3, because they report changes to non-BI entities in the versions examined. I only compared techniques with respect to the binary incompatible changes each makes predictions for.

D.3 Comparison with Schäfer et al.

Schäfer et al. made the results of their evaluation using the STRUTS v1.1.0 to v1.2.4 migration available online; however, the link provided in their paper has not worked since 2012.[7] I was able to obtain the results of their evaluation through the Master's thesis of the student involved in the project, Jonas, as a listing of their results is included as an appendix [Jonas, 2007].[8] Schäfer et al. [2008] report change rules their approach detects, describing how a library's API has changed between versions; I compare these detected change rules against those API changes recorded in the Corpus.

[7] I first tried to access their data in June 2012, the data may have been removed prior to that.

[8] I appreciate Jonas taking the trouble to e-mail me a copy of his thesis.

The data presented in Table D.5 and Table D.6 was manually transcribed from the thesis of Jonas [2007], with the exception that I omit a prefix he used to indicate the source of each change rule. The package names and constructor invocation notations used by Schäfer et al. are not consistent with their recording in the Corpus, but are similar enough that the reader should not be confused.

There are some limitations in the provided data. First, while Jonas reports his approach as detecting 75 changes, his thesis only records the 66 changes deemed to be correct. Secondly, there is a small discrepancy in the results reported in [Jonas, 2007], and those reported in [Schäfer et al., 2008]: Jonas reports a total of 75 change rules identified, while Schäfer et al. reports 77. Schäfer et al. report their approach as having 85.7% precision in identifying API usage changes in the STRUTS v1.1.0 to v1.2.4 library migration, and while they report that 20 change rules were missed (i.e., false negatives), they do not report the recall of their approach; Jonas reports 88.0% precision, with 61.1% recall.

Jonas [2007] is the only currently available source of data describing what the approach of Schäfer et al. does in fact detect; thus, I will try to be cognizant of these limitations and discrepancies in discussing this work.

D.3.1 Changes on which Schäfer et al. and the Corpus Agree

Table D.5 lists the API changes detected by Schäfer et al. which the Corpus identify as BI changes, and for which the recommended replacement are in agreement.

Table D.5: Schäfer et al. API Changes in Agreement with the Corpus. (All in org.apache.struts)

Original	Replacement
action.Action.perform(action.ActionMapping,action.ActionForm ,javax.servlet.http.HttpServletRequest,	action.Action.execute(action.ActionMapping,action.ActionForm ,javax.servlet.http.HttpServletRequest,

Continued on next page

Table D.5 — continued from previous page

Original	Replacement
action.Action.perform(action.ActionMapping,action.ActionForm	action.Action.execute(action.ActionMapping,action.ActionForm
,javax.servlet.http.HttpServletRequest,	,javax.servlet.http.HttpServletRequest,
action.ActionMessages.ActionMessageItem(java.util.List,**int**)	action.ActionMessages.ActionMessageItem(java.util.List,**int**,
	String)
tiles.actions.TilesAction.perform(tiles.ComponentContext,	tiles.actions.TilesAction.execute(tiles.ComponentContext,
action.ActionMapping,action.ActionForm,	action.ActionMapping,action.ActionForm,
tiles.TilesUtil.applicationClass(String)	util.RequestUtils.applicationClass(String)
tiles.UntyppedAttribute	tiles.UntypedAttribute
tiles.UntyppedAttribute(Object)	tiles.UntypedAttribute(Object)
tiles.UntyppedAttribute(Object,String)	tiles.UntypedAttribute(Object,String)
tiles.UntyppedAttribute.setRole(String)	tiles.UntypedAttribute.setRole(String)
validator.Resources.initValidator(String,Object,javax.servlet.	validator.Resources.initValidator(String,Object,javax.servlet.
ServletContext,javax.servlet.http.HttpServletRequest,	ServletContext,javax.servlet.http.HttpServletRequest,

There are several cases in which a single change is duplicated: this is due to how Schäfer et al. record change data in their analysis. While their approach is similar to that of Kim et al. [2007], they do not seem merge duplicate or related rules. As a result, there is a degree of over-reporting correct results, where the actual number of API changes they correct identify adaptations for is less than the number of change rules collected.

As an example, the org.apache.struts.tiles.UntyppedAttribute class was renamed to UntypedAttribute in the next API version (to fix a spelling mistake): Schäfer et al. report a total of four change rules: one describing the class rename, and three describing its component methods being remapped to their equivalent in the new class. Similarly the org.apache.struts.action.Action.perform() method has two different change rules describing its change between versions.

Where are Schäfer et al. applying call analysis? It should be noted that the approach of Schäfer et al. applies call analysis on already migrated instantiations that use the library's API; in the absence of such instantiations, they assume that a provided test suite will contain sufficient examples. However, as reported in Section 5.5.2.2, only two of the API changes the Corpus records for the STRUTS 1.1.0 to 1.2.4 migration are referenced anywhere in the STRUTS 1.1.0 test suite. They are:

- the org.apache.struts.config.ApplicationConfig class, which is used in several test cases but is not tested directly; and,
- the org.apache.struts.util.RequestUtils.selectApplication() method, which is also referenced in several test cases.

At the time of the analysis in Section 5.5.2.2, I did not have access to the results of Schäfer et al.; I could only speculate on what portion of their identified changes must have originated from applying call-analysis to the migrated STRUTS test suite.

However, if we rely on the reported results of Jonas [2007] in Table D.5, it is likely that *none* of the binary incompatible API changes Schäfer et al. identified rules for can be found in the STRUTS test suite.[9] It is most likely that all of the correct change rules identified came from applying call-analysis to the STRUTS library implementation itself. This has important implications for their assumption, outlined in Section 5.5.2.2, that test suites can provide enough usage examples to make their approach viable in identifying adaptations for API migration; this was also offered as a key differentiator between their work, and that of Dagenais and Robillard [2011].

D.3.2 Non-BI Changes Considered by Schäfer et al.

Table D.6 lists those changes reported by Schäfer et al. in STRUTS that do not reflect BI changes to the API. I characterize their recorded changes as comprising the following cases.

[9]This was verified by using the tool support in the Eclipse IDE to look for references in the test suite code.

Table D.6: Non-BI Changes Recorded by Schäfer et al.. (All in org.apache.struts)

Original	Replacement
action.Action.saveErrors(javax.servlet.http.HttpServletRequest, action.ActionErrors)	action.Action.saveErrors(javax.servlet.http.HttpServletRequest, action.ActionMessages)
action.ActionError(String)	action.ActionMessage(String)
action.ActionError(String,Object)	action.ActionMessage(String,Object)
action.ActionErrors()	action.ActionMessages()
action.ActionErrors.add(String,action.ActionError)	action.ActionMessages.add(String,action.ActionMessage)
action.ActionMapping.findException(Class)	config.ActionConfig.findException(Class)
action.ActionServlet.getRequestProcessor(config. ModuleConfig)	action.ActionServlet.getProcessorForModule(config. ModuleConfig)
action.ActionServlet.parseModuleConfigFile(String,String, config.ModuleConfig,commons.digester.Digester,String)	action.ActionServlet.parseModuleConfigFile(commons. digester.Digester,String)
action.DynaActionForm.reset(action.ActionMapping,javax. servlet.http.HttpServletRequest)	action.ActionForm.reset(action.ActionMapping,javax.servlet. http.HttpServletRequest)
action.DynaActionFormClass.createDynaActionFormClass(config.FormBeanConfig)	config.FormBeanConfig.getDynaActionFormClass()
action.ExceptionHandler.storeException(javax.servlet.http. HttpServletRequest,String,action.ActionError,action. ActionForward,String)	action.ExceptionHandler.storeException(javax.servlet.http. HttpServletRequest,String,action.ActionMessage,action. ActionForward,String)
config.impl.ModuleConfigImpl.addActionConfig(config. ActionConfig)	config.ModuleConfig.addActionConfig(config.ActionConfig)
config.impl.ModuleConfigImpl.addFormBeanConfig(config. FormBeanConfig)	config.ModuleConfig.addFormBeanConfig(config. FormBeanConfig)

Continued on next page

Table D.6 — continued from previous page

Original	Replacement
config.impl.ModuleConfigImpl.addForwardConfig(config. ForwardConfig)	truts.config.ModuleConfig.addForwardConfig(config. ForwardConfig)
config.impl.ModuleConfigImpl.findActionConfig(String)	config.ModuleConfig.findActionConfig(String)
config.impl.ModuleConfigImpl.findFormBeanConfig(String)	config.ModuleConfig.findFormBeanConfig(String)
config.impl.ModuleConfigImpl.findForwardConfig(String)	config.ModuleConfig.findForwardConfig(String)
config.impl.ModuleConfigImpl.getControllerConfig()	config.ModuleConfig.getControllerConfig()
config.impl.ModuleConfigImpl.getPrefix()	config.ModuleConfig.getPrefix()
config.impl.ModuleConfigImpl.setControllerConfig(config. ControllerConfig)	config.ModuleConfig.setControllerConfig(config. ControllerConfig)
taglib.html.JavascriptValidatorTag.getJavascriptEnd()	taglib.html.JavascriptValidatorTag.renderJavascript()
taglib.html.JavascriptValidatorTag.getStartElement()	taglib.html.JavascriptValidatorTag.renderStartElement()
tiles.DefinitionsUtil.getDefinitionsFactory(javax.servlet. ServletContext)	tiles.TilesUtil.getDefinitionsFactory(javax.servlet. ServletRequest,javax.servlet.ServletContext)
tiles.TilesUtilImpl.doInclude(String,javax.servlet.jsp. PageContext)	tiles.TilesUtilImpl.doInclude(String,javax.servlet.jsp. PageContext)
util.ModuleException.getError()	util.ModuleException.getActionMessage()
util.RequestUtils.computeParameters(javax.servlet.jsp. PageContext,String,String,String,String,String,String,String, **boolean**)	taglib.TagUtils.computeParameters(javax.servlet.jsp. PageContext,String,String,String,String,String,String,String, **boolean**)
util.RequestUtils.computeURL(javax.servlet.jsp.PageContext, String,String,String,java.util.Map,String,**boolean**)	taglib.TagUtils.computeURLWithCharEncoding(javax.servlet. jsp.PageContext,String,String,String,String,String,java.util.Map ,String,**boolean**,**boolean**)

Continued on next page

Table D.6 — continued from previous page

Original	Replacement
util.RequestUtils.computeURL(javax.servlet.jsp.PageContext, String,String,String,String,java.util.Map,String,**boolean**)	taglib.TagUtils.computeURL(javax.servlet.jsp.PageContext, String,String,String,String,String,java.util.Map,String,**boolean**)
util.RequestUtils.computeURL(javax.servlet.jsp.PageContext, String,String,String,String,java.util.Map,String,**boolean**, **boolean**)	taglib.TagUtils.computeURL(javax.servlet.jsp.PageContext, String,String,String,String,String,java.util.Map,String,**boolean**, **boolean**)
util.RequestUtils.encodeURL(String)	taglib.TagUtils.encodeURL(String,String)
util.RequestUtils.forwardURL(javax.servlet.http. HttpServletRequest,config.ForwardConfig)	util.RequestUtils.forwardURL(javax.servlet.http. HttpServletRequest,config.ForwardConfig,config.ModuleConfig)
util.RequestUtils.getActionMappingName(String)	taglib.TagUtils.getActionMappingName(String)
util.RequestUtils.getActionMessages(javax.servlet.jsp. PageContext,String)	taglib.TagUtils.getActionMessages(javax.servlet.jsp. PageContext,String)
util.RequestUtils.getModuleCon_x000C_g(javax.servlet.http. HttpServletRequest,javax.servlet.ServletContext)	util.ModuleUtils.getModuleCon
util.RequestUtils.getModuleCon_x000C_g(javax.servlet.jsp. PageContext)	taglib.TagUtils.getModuleCon
util.RequestUtils.getModuleName(String,javax.servlet. ServletContext)	util.ModuleUtils.getModuleName(String,javax.servlet. ServletContext)
util.RequestUtils.getRequestModuleConfig(javax.servlet.http. HttpServletRequest)	util.ModuleUtils.getModuleConfig(javax.servlet.http. HttpServletRequest)
util.RequestUtils.getScope(String)	taglib.TagUtils.getScope(String)
util.RequestUtils.isXhtml(javax.servlet.jsp.PageContext)	taglib.TagUtils.isXhtml(javax.servlet.jsp.PageContext)

Continued on next page

Table D.6 — continued from previous page

Original	Replacement
util.RequestUtils.lookup(javax.servlet.jsp.PageContext,String, String)	taglib.TagUtils.lookup(javax.servlet.jsp.PageContext,String, String)
util.RequestUtils.lookup(javax.servlet.jsp.PageContext,String, String,String)	taglib.TagUtils.lookup(javax.servlet.jsp.PageContext,String, String,String)
util.RequestUtils.message(javax.servlet.jsp.PageContext,String ,String,String)	taglib.TagUtils.message(javax.servlet.jsp.PageContext,String, String,String)
util.RequestUtils.message(javax.servlet.jsp.PageContext,String ,String,String,Object[])	taglib.TagUtils.message(javax.servlet.jsp.PageContext,String, String,String,Object[])
util.RequestUtils.present(javax.servlet.jsp.PageContext,String, String,String)	taglib.TagUtils.present(javax.servlet.jsp.PageContext,String, String,String)
util.RequestUtils.retrieveUserLocale(javax.servlet.jsp. PageContext,String)	taglib.TagUtils.getUserLocale(javax.servlet.jsp.PageContext, String)
util.RequestUtils.saveException(javax.servlet.jsp.PageContext, Throwable)	taglib.TagUtils.saveException(javax.servlet.jsp.PageContext, Throwable)
util.RequestUtils.selectModule(javax.servlet.http. HttpServletRequest,javax.servlet.ServletContext)	util.ModuleUtils.selectModule(javax.servlet.http. HttpServletRequest,javax.servlet.ServletContext)
util.RequestUtils.selectModule(String,javax.servlet.http. HttpServletRequest,javax.servlet.ServletContext)	util.ModuleUtils.selectModule(String,javax.servlet.http. HttpServletRequest,javax.servlet.ServletContext)
util.ResponseUtils.filter(String)	taglib.TagUtils.filter(String)
util.ResponseUtils.write(javax.servlet.jsp.PageContext,String)	taglib.TagUtils.write(javax.servlet.jsp.PageContext,String)
util.ResponseUtils.writePrevious(javax.servlet.jsp.PageContext ,String)	taglib.TagUtils.writePrevious(javax.servlet.jsp.PageContext, String)

Continued on next page

Table D.6 — continued from previous page

Original	Replacement
validator.DynaValidatorForm.validate(action.ActionMapping, javax.servlet.http.HttpServletRequest)	validator.DynaValidatorForm.getValidationKey(action. ActionMapping,javax.servlet.http.HttpServletRequest)
validator.Resources.getActionError(javax.servlet.http. HttpServletRequest,commons.validator.ValidatorAction, validator.Resources.getLocale(javax.servlet.http. HttpServletRequest)	validator.Resources.getActionMessage(javax.servlet.http. HttpServletRequest,commons.validator.ValidatorAction, util.RequestUtils.getUserLocale(javax.servlet.http. HttpServletRequest,String)
validator.Resources.initValidator(String,Object,javax.servlet. ServletContext,javax.servlet.http.HttpServletRequest, validator.ValidatorForm.validate(action.ActionMapping,javax. servlet.http.HttpServletRequest)	validator.Resources.initValidator(String,Object,javax.servlet. ServletContext,javax.servlet.http.HttpServletRequest, validator.ValidatorForm.getValidationKey(action.ActionMapping ,javax.servlet.http.HttpServletRequest)

- *Classes or methods marked as deprecated.* Several of the changes involve methods or classes marked as deprecated, but not removed in STRUTS v1.2.4. For example, the struts.action.ActionError class is marked as deprecated in v1.2.4, and its replacement is identified as its parent class; Schäfer et al. reports usage changes for two of org.apache.struts.action.ActionError's six constructors that reflect this (the changes reported on methods belonging to the org.apache.struts.util.RequestUtils class belong to this category as well). As with Section D.2.3, these entities were present in all versions of the STRUTS library that the Corpus examines; they seem to be removed sometime prior to the final v1.3.10 release.

- *Calling methods on parent classes/interfaces.* A number of usage changes that Schäfer et al. report reflect modifying code to invoke methods on a parent class, or implemented interface. For example, the authors report a usage change involving the org.apache.struts.action.DynaActionForm.reset() method, where the replace-

ment is simply the same method in DynaActionForm's parent class. Similarly, a number of methods belonging to the struts.config.impl.ModuleConfigImpl class are reported as being changed to the org.apache.struts.config.ModuleConfig interface, but ModuleConfigImpl is simply a concrete realization of the ModuleConfig interface.

- *Methods moved into a superclass.* A few changes reflect the *pull-up* refactoring, where a method is moved into its superclass, which is a backwards compatible API change. One is the org.apache.struts.action.ActionMapping.findException() method, which is moved into its parent, org.apache.struts.action.ActionConfig, in the next version. This is a backwards compatible change, as this method is still accessible from ActionMapping (the same issue occurs in Section D.2.3).

In all of these cases, the original functionality remains in the library, or is moved into a superclass. The behaviour of this functionality is unchanged. The Corpus does not report these as binary incompatible API changes.

D.3.3 Changes on which the Corpus and Schäfer et al. Disagree

There are no cases in the data of Schäfer et al. in which a BI API change detailed in the Corpus disagrees with an identified change by Schäfer et al..

D.3.4 Comparison with Schäfer et al. [2008]: Summary

Table D.7: Comparison of Schäfer et al. to the Corpus for STRUTS 1.1.0 to 1.2.4.

Changes Identified	BI Changes			Non-BI Changes
	Agrees	Duplicates	Disagrees	
66	6	4	0	56

Table D.7 summarizes the comparison of the approach of Schäfer et al. for identifying API change rules to those of the Corpus for the STRUTS 1.1.0 to 1.2.4 migration, using the data provided in

Jonas [2007]. Of the 66 changes they identify, 10 are in agreement with the Corpus, but 4 of those changes are effectively duplicates of an existing change rule their approach had earlier identified. The remaining 56 change rules detected reflect deprecations in the API where functionality was still present, or trivial implementation changes where methods are called on parent classes instead of a concrete subclass.

While their work asserts that the test suite associated with the STRUTS library migration would contain useful examples for redressing API change, none of the correctly recommend adaptations for BI API changes could have originated from the STRUTS test suite, and only two of the BI changes identified in the Corpus are referenced in any test suite code.

D.4 Comparison with Wu et al.

Wu et al. have made the evaluation of their AURA tool on the STRUTS v1.1.0 to v1.2.4 migration available online[10]; the comparisons in this section are taken from this dataset. Like Schäfer et al. [2008], AURA reports change rules it detects as having occurred in the library's API between versions; I compare these detected rules against those API changes recorded in the Corpus.

The data presented in Table D.8, Table D.9, Table D.10 and Table D.12 is presented as recorded in the provided data for AURA, with the exception that I omit their tokenization of the entities, and a "sound" and "distance" tag whose value is not explained in the paper.[11] Wu et al. report AURA as having 96.56% precision in detecting change rules; while they do not report recall directly, they state AURA improving in recall over the work of Schäfer et al. by 95.49%.

Wu et al. do not report the total number of change rules that AURA detects for the STRUTS v1.1.0 to v1.2.4 migration: instead, they report AURA detecting 129 correct change rules. Based on their reported precision, this suggests they identified approximately 134 change rules in total.[12]

[10] http://www.ptidej.net/downloads/replications/icse10b

[11] The data analyzed is from the configuration of AURA that tries to detect if methods are being deleted, as presented in [Wu et al., 2010, Table 3, p.331].

[12] There is some confusion here, as neither 133 or 134 change rules would result in the precision they report. $129/0.9656 = 133.60$, but 134 total rules would yield a precision of 96.27%, while 133 total rules yields 96.99%.

I report AURA has discovering 135 change rules, based on the results in their provided data. Neither their paper nor their data indicates those change rules they considered to be wrong.

D.4.1 Changes on which Wu et al. and the Corpus Agree

Table D.8 lists the API changes detected by AURA which the Corpus identify as BI changes, and for which the recommended replacement are correct in each dataset.

Table D.8: Wu et al. Matched API Changes. (All in org.apache.struts)

Original	Replacement
action.ActionMessages.ActionMessageItem.	action.ActionMessages.ActionMessageItem.
ActionMessageItem	ActionMessageItem
config.ConfigHelper.getLocale	util.RequestUtils.getUserLocale
config.ConfigHelperInterface.getActionErrors	config.ConfigHelperInterface.getActionMessages
config.FormBeanConfig.setModuleConfig	none
taglib.html.FormTag.getName	taglib.html.FormTag.getBeanName
taglib.html.FrameTag.getMarginheight	taglib.html.FrameTag.getMarginheight
taglib.html.FrameTag.getMarginwidth	taglib.html.FrameTag.getMarginwidth
taglib.html.FrameTag.setMarginheight	taglib.html.FrameTag.setMarginheight
taglib.html.FrameTag.setMarginwidth	taglib.html.FrameTag.setMarginwidth
tiles.UntyppedAttribute.getRole	tiles.UntypedAttribute.getRole
tiles.UntyppedAttribute.getValue	tiles.UntypedAttribute.getValue
tiles.UntyppedAttribute.setRole	tiles.UntypedAttribute.setRole
tiles.UntyppedAttribute.setValue	tiles.UntypedAttribute.setValue
tiles.UntyppedAttribute.toString	tiles.UntypedAttribute.toString
tiles.UntyppedAttribute.UntyppedAttribute	tiles.UntypedAttribute.UntypedAttribute
tiles.UntyppedAttribute.UntyppedAttribute	tiles.UntypedAttribute.UntypedAttribute

Continued on next page

Table D.8 — continued from previous page

Original	Replacement
validator.FieldChecks.validateByte	validator.FieldChecks.validateByte
validator.FieldChecks.validateCreditCard	validator.FieldChecks.validateCreditCard
validator.FieldChecks.validateDate	validator.FieldChecks.validateDate
validator.FieldChecks.validateDouble	validator.FieldChecks.validateDouble
validator.FieldChecks.validateDoubleRange	validator.FieldChecks.validateDoubleRange
validator.FieldChecks.validateEmail	validator.FieldChecks.validateEmail
validator.FieldChecks.validateFloat	validator.FieldChecks.validateFloat
validator.FieldChecks.validateFloatRange	validator.FieldChecks.validateFloatRange
validator.FieldChecks.validateInteger	validator.FieldChecks.validateInteger
validator.FieldChecks.validateIntRange	validator.FieldChecks.validateIntRange
validator.FieldChecks.validateLong	validator.FieldChecks.validateLong
validator.FieldChecks.validateMask	validator.FieldChecks.validateMask
validator.FieldChecks.validateMaxLength	validator.FieldChecks.validateMaxLength
validator.FieldChecks.validateMinLength	validator.FieldChecks.validateMinLength
validator.FieldChecks.validateRequired	validator.FieldChecks.validateRequired
validator.FieldChecks.validateRequiredIf	validator.FieldChecks.validateRequiredIf
validator.FieldChecks.validateShort	validator.FieldChecks.validateShort
validator.Resources.initValidator	validator.Resources.initValidator

For the most part, the recommended replacements of AURA match those contained in the Corpus; however, there are a few cases in which the recommendation is different, but still valid.

- A class name in STRUTS v1.1.0 had a spelling error: org.apache.struts.tiles.UntyppedAttribute has an extra 'p' in "Untyped", and this is fixed in v1.2.4. The Corpus reports this as a class-level rename refactoring; the AURA tool reports this as

7 method level transformations, as the tool does not currently handle class-level of field-level changes. This however should still be considered correct.

- AURA reports the org.apache.struts.config.FormBeanConfig.setModuleConfig() method as being deleted without replacement, while the Corpus notes that using javax.util.HttpServletRequest.setAttribute() method can be used to provide equivalent functionality, with Globals.MODULE_KEY being supplied as the first parameter.

  This replacement though is not within the STRUTS library, but is part of the JAVA Enterprise Edition platform on which STRUTS is based. AURA does not consider replacements from outside the library; thus, its treatment of this method as being deleted is correct *in that context*.

- The Corpus reports that several methods in org.apache.struts.taglib.html.FormTag were eliminated as part of STRUTS's transition to using JAVABEANS for configuring objects. Instead of directly accessing field information using getter/setter methods, callers instead would need to use the javax.servlet.jsp.tagext.TagSupport.getValue() or javax.servlet.jsp.tagext.TagSupport.setValue() methods, where TagSupport is the superclass of FORMTAG. Callers would then be able to dynamically access properties by supplying the attribute they wished to retrieve (e.g., "name"), rather then needing to specify a specific method invocation on an object.

  However, in the case of the FormTag.getName() method, AURA found that a new method introduced in version 1.2.4, FormTag.getBeanName(), was a replacement. This is also valid: while five getter/setter methods were eliminated in FormTag as part of this movement to JAVABEANS, the developers left in a direct replacement for the getName() method that I did not notice. This is an alternative for accessing the functionality, and should be considered correct.[13]

[13]In identifying this alternative replacement, I also discovered that three fields in FormTag that I had originally marked in the Corpus as being provided through a configuration file (specifically, the FormTag.name, FormTag.type,

D.4.2 Non-Library Changes Considered by Wu et al.

Table D.9 lists changes that AURA detects which are not to the STRUTS library, but rather are to the STRUTS-TILES framework (see also Section D.2.2), while Table D.10 list changes made to test suite and example code that was provided with the 1.1.0 release. Because such changes are not part of the core STRUTS library, I excluded them from analysis, and cannot speak to whether these represent binary incompatibilities in STRUTS-TILES framework, the example code, or the test suite. The Corpus only considered changes to the STRUTS API itself.

Table D.9: Wu et al. Changes External to the STRUTS API. (All in org.apache.struts)

Original	Replacement
webapp.example.CheckLogonTag.getName	webapp.example.LinkSubscriptionTag.getName
webapp.example.CheckLogonTag.getPage	webapp.example.LinkSubscriptionTag.getPage
webapp.example.CheckLogonTag.setName	webapp.example.LinkUserTag.setName
webapp.example.CheckLogonTag.setPage	webapp.example.LinkUserTag.setPage
webapp.example.LogonAction.getUser	webapp.example.LogonAction.getUser
webapp.example.memory.MemoryDatabasePlugIn.	none
setupCache	
webapp.example.memory.	webapp.example.memory.
MemorySubscriptionCreationFactory.	MemorySubscriptionCreationFactory.
MemorySubscriptionCreationFactory	MemorySubscriptionCreationFactory
webapp.tiles.channel.SelectChannelAction.perform	webapp.tiles.channel.SelectChannelAction.execute
webapp.tiles.dyn.RetrievePortalAction.perform	webapp.tiles.dyn.RetrievePortalAction.execute
webapp.tiles.dyn.SetPortalPrefsAction.perform	webapp.tiles.dyn.SetPortalPrefsAction.execute
webapp.tiles.invoice.EditInvoiceAction.perform	webapp.tiles.invoice.EditInvoiceAction.execute

Continued on next page

and FormTag.scope fields) *did* have equivalent fields in the new API version in that class. These were corrected, and are part of the 2.0 Library Migration Corpus.

Table D.9 — continued from previous page

Original	Replacement
webapp.tiles.lang.SelectLocaleAction.perform	webapp.tiles.lang.SelectLocaleAction.execute
webapp.tiles.portal.UserMenuAction.perform	webapp.tiles.portal.UserMenuAction.execute
webapp.tiles.portal.UserMenuSettingsAction.perform	webapp.tiles.portal.UserMenuSettingsAction.execute
webapp.tiles.portal.UserPortalAction.perform	webapp.tiles.portal.UserPortalAction.execute
webapp.tiles.portal.UserPortalSettingsAction.perform	webapp.tiles.portal.UserPortalSettingsAction.execute
webapp.tiles.rss.Channels.perform	webapp.tiles.rss.Channels.execute
webapp.tiles.rss.RssChannelsAction.perform	webapp.tiles.rss.RssChannelsAction.execute
webapp.tiles.skin.LayoutSettingsAction.perform	webapp.tiles.skin.LayoutSettingsAction.execute
webapp.tiles.skin.LayoutSwitchAction.perform	webapp.tiles.skin.LayoutSwitchAction.execute
webapp.tiles.skin.SimpleSwitchLayoutAction.perform	webapp.tiles.skin.SimpleSwitchLayoutAction.execute
webapp.tiles.template.DynTemplateAction.perform	webapp.tiles.template.DynTemplateAction.execute
webapp.tiles.test.TestActionTileAction.perform	webapp.tiles.test.TestActionTileAction.execute
webapp.tiles.tutorial.ForwardExampleAction.perform	webapp.tiles.tutorial.ForwardExampleAction.execute

Table D.10: Wu et al. Changes to the STRUTS test suite. (All in org.apache.struts)

Original	Replacement
taglib.bean.TestCookieTag.beginCookieTagName	taglib.bean.TestHeaderTag.beginHeaderTagName
taglib.bean.TestCookieTag.beginCookieTagNameMultiple	taglib.bean.TestHeaderTag.beginHeaderTagNameMultiple
taglib.bean.TestCookieTag.beginCookieTagNameValue	taglib.bean.TestHeaderTag.beginHeaderTagNameValue
taglib.bean.TestCookieTag.endCookieTagName	taglib.bean.TestWriteTag.endWriteTagName
taglib.bean.TestCookieTag.endCookieTagNameMultiple	taglib.bean.TestHeaderTag.endHeaderTagNameMultiple
taglib.bean.TestCookieTag.endCookieTagNameValue	taglib.bean.TestHeaderTag.endHeaderTagNameValue
taglib.bean.TestCookieTag.testCookieTagName	taglib.bean.TestCookieTag.testDummy

Continued on next page

Table D.10 — continued from previous page

Original	Replacement
taglib.bean.TestCookieTag.testCookieTagNameMultiple	taglib.bean.TestHeaderTag.testHeaderTagNameMultiple
taglib.bean.TestCookieTag.testCookieTagNameValue	taglib.bean.TestHeaderTag.testHeaderTagNameValue
taglib.bean.TestIncludeTag.endIncludeTagForward	taglib.bean.TestStrutsTag.endStrutsTagForward
taglib.bean.TestIncludeTag.endIncludeTagHref	taglib.bean.TestIncludeTag.testIncludeTagHref
taglib.bean.TestIncludeTag.endIncludeTagPage	taglib.bean.TestIncludeTag.testIncludeTagPage
taglib.bean.TestIncludeTag.formatAndTest	taglib.bean.TestDefineTag.formatAndTest
taglib.logic.TestEqualTag.beginCookieStringEquals	taglib.logic.TestEqualTag.beginCookieStringNotEquals
taglib.logic.TestEqualTag.testCookieStringEquals	taglib.logic.TestEqualTag.testCookieStringNotEquals
taglib.logic.TestGreaterEqualTag.beginCookieGreaterEqual	taglib.logic.TestGreaterEqualTag.beginHeaderGreaterEqual
taglib.logic.TestGreaterEqualTag.testCookieGreaterEqual	taglib.logic.TestGreaterEqualTag.testHeaderGreaterEqual
taglib.logic.TestGreaterThanTag.beginCookieGreaterThan	taglib.logic.TestGreaterThanTag.beginHeaderGreaterThan
taglib.logic.TestGreaterThanTag.testCookieGreaterThan	taglib.logic.TestGreaterThanTag.testParameterGreaterThan
taglib.logic.TestNotEqualTag.beginCookieStringEquals	taglib.logic.TestNotEqualTag.beginCookieStringNotEquals
taglib.logic.TestNotEqualTag.testCookieStringEquals	taglib.logic.TestNotEqualTag.testCookieStringNotEquals
taglib.logic.TestNotPresentTag.beginCookiePresent	taglib.logic.TestNotPresentTag.beginHeaderPresent
taglib.logic.TestNotPresentTag.testCookiePresent	taglib.logic.TestNotPresentTag.testCookieNotPresent
taglib.logic.TestPresentTag.beginCookiePresent	taglib.logic.TestPresentTag.beginParameterPresent
taglib.logic.TestPresentTag.testCookiePresent	taglib.logic.TestPresentTag.testCookieNotPresent
util.RequestUtils.retrieveMessageResources	taglib.TagUtils.getInstance

D.4.3 Non-API Changes Considered by Wu et al.

Table D.11 lists changes that AURA detects which are to the STRUTS library implementation, but not to the API. The detected changes comprise two broad categories.

Table D.11: Wu et al. Non-API Changes. (All in org.apache.struts)

Original	Replacement
Modifications to method with private visibility.	
action.ActionServlet.parseModuleConfigFile	action.ActionServlet.parseModuleConfigFile
taglib.html.JavascriptValidatorTag.getNextVar	taglib.html.JavascriptValidatorTag.renderJavascript
taglib.html.JavascriptValidatorTag.getStartElement	taglib.html.JavascriptValidatorTag.renderStartElement
taglib.html.JavascriptValidatorTag.replaceChar	taglib.html.JavascriptValidatorTag.renderStartElement
Changes identified in methods belonging to final, inner class with package visibility.	
config.AddDataSourcePropertyRule.begin	config.AddDataSourcePropertyRule.begin
config.PlugInSetPropertyRule.begin	config.PlugInSetPropertyRule.begin
config.SetActionMappingClassRule.begin	config.SetActionMappingClassRule.begin
Pull-up refactorings.	
action.ActionMapping.findException	config.ActionConfig.findException
validator.DynaValidatorActionForm.validate	validator.DynaValidatorForm.validate
validator.ValidatorActionForm.validate	validator.ValidatorForm.validate
Incorrectly identified replacements.	
tiles.EmptyIterator.hasNext	taglib.tiles.InsertTag.getFlush
tiles.EmptyIterator.next	tiles.AttributeDefinition.getValue
tiles.EmptyIterator.remove	tiles.TilesPlugin.destroy

1. *Changes identified to entities with **private** or **package** visibility.* One of the purposes of establishing an API is to provide an abstraction layer which other software systems can use without being affected by changes to the internal implemen-

```
1  public class ConfigRuleSet extends RuleSetBase {
2  //Code elided
3    /**
4     * Class that records the name and value of a configuration property to be
5     * used in configuring a <code>PlugIn</code> instance when instantiated.
6     */
7
8    final class PlugInSetPropertyRule extends Rule {
9
10     public PlugInSetPropertyRule() {
11       super();
12     }
13
14     public void begin(Attributes attributes) throws Exception {
15       PlugInConfig plugInConfig = (PlugInConfig) digester.peek();
16       plugInConfig.addProperty(attributes.getValue("property"),
17                                attributes.getValue("value"));
18     }
19
20   }
21 //Code elided
22 }
```

Listing D.8: The PlugInSetPropertyRule class in the ConfigRuleSet class, v1.1.0

tation [Bloch, 2006]. Entities marked as **private** are excluded from the API, and are expected to change freely as developers update the implementation of their libraries. It is not clear that identifying such changes helps developers who are seeking to migrate their code in response to API changes.

In some cases, the changes may appear to be relevant, particularly when they involve methods on inner or nested classes that do not declare a visibility. For example, the org.apache.struts.config.ConfigRuleSet class contains a series of inner classes, such as PlugInSetPropertyRule, whose implementation is shown in Listing D.8. Inner classes, and their methods, when marked as **public** or **protected** are normally accessible through the API, so at first glance it might seem that this is a relevant change to track. But PlugInSetPropertyRule does not declare a visibility; in so doing, it assumes a default visibility of "package-protected" in JAVA, meaning that this class is accessible only to other entities in the same package (i.e., other entities in

the org.apache.struts.config package in STRUTS).[14] By definition, the class and its methods will not be accessible through the API.

2. *Changes that do not result in a binary incompatibility.* AURA identified a series of changes that were caused by pull-up refactorings, where methods were moved into a parent super class. For example, the findException() method was moved from the org.apache.struts.action.ActionMapping class into its parent, the org.apache.struts.config.ActionConfig class. As noted earlier in Section D.2.3 and Section D.3.2, such changes are binary compatible, as the findException() method is still accessible on the ActionMapping class, as it now inherits this from ActionConfig.

The Corpus rightly does not record these as changes to the STRUTS API. While AURA's identification of these changes occurring in the library's implementation are correct, they are of little use in recommending adaptive changes for API migration.

D.4.3.1 Non-API Changes that Are Wrong

Table D.11 also shows changes AURA identified to three methods in the org.apache.struts.tiles.EmptyIterator class: hasNext(), next(), and remove(), the implementations of which are shown in Listing D.9. This inner class is deleted in the next STRUTS release; however, because it is an inner class with the implied default visibility of "package-protected", it is not part of the API, cannot be accessed in external code, and rightly does not form part of the Corpus. The reason I discuss this change further is that the replacements AURA identified for EmptyIterator's methods are incorrect in an unusual fashion.

As shown in Listing D.9, the EmptyIterator class was used to provide an Iterator object that was empty; it contained nothing to iterate over. In examining its usage in ComponentContext.getAttributeNames(), it seems that this inner class was created as an alternative to returning a **null** value when requests were made for non-existent attribute names.

[14] http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

```
1  public class ComponentContext implements Serializable
2  {
3  /*Code elided*/
4     /**
5      * EmptyIterator over component attributes.
6      */
7     private static Iterator EMPTY_ITERATOR = new EmptyIterator();
8  /*Code elided*/
9     /**
10    * Get names of all attributes.
11    * @return <{Object}>
12    */
13    public Iterator getAttributeNames()
14    {
15    if( attributes == null )
16    return EMPTY_ITERATOR;
17    return attributes.keySet().iterator();
18    }
19 /*Code elided*/
20    class EmptyIterator implements Iterator {
21
22       public boolean hasNext()
23       {
24        return false;
25       }
26
27       public Object next()
28       {
29       throw new java.util.NoSuchElementException();
30       }
31
32       public void remove()
33       {
34       throw new UnsupportedOperationException();
35       }
36     } // end inner class
37 }
```

Listing D.9: The EmptyIterator inner class in the ComponentContext class, v1.1.0

Listing D.11 shows the relevant implementation of ComponentContext in the next version of STRUTS; the getAttributeNames() method now uses an empty iterator provided by the JAVA API through a static field, java.lang.Collections.EMPTY_LIST. This allows the STRUTS developers to remove unnecessary code from their library.

It is worth noting two things at this point. First, call analysis (which AURA implements) can detect this change: as Listing D.9 shows, the EmptyIterator class is referenced in a statically

```
1  public class ComponentContext implements Serializable
2  {
3     \\Code elided
4     /**
5      * EmptyIterator over component attributes.
6      */
7     private static Iterator EMPTY_ITERATOR = new EmptyIterator();
8     \\Code elided
9     /**
10     * Get names of all attributes.
11     * @return <{Object}>
12     */
13    public Iterator getAttributeNames()
14    {
15    if( attributes == null )
16    return EMPTY_ITERATOR;
17    return attributes.keySet().iterator();
18    }
19    \\Code elided
20 }
```

Listing D.10: The EmptyIterator inner class in the ComponentContext class, v1.1.0

```
1  public class ComponentContext implements Serializable
2  {
3     \\Code elided
4     /**
5      * EmptyIterator over component attributes.
6      */
7     private static Iterator EMPTY_ITERATOR = new EmptyIterator();
8     \\Code elided
9     /**
10     * Get names of all attributes.
11     * @return <{Object}>
12     */
13    public Iterator getAttributeNames()
14    {
15    if( attributes == null )
16    return EMPTY_ITERATOR;
17    return attributes.keySet().iterator();
18    }
19    \\Code elided
20 }
```

Listing D.11: The getAttributeNames() method from the ComponentContext class, v1.2.4

defined field, and this field is used within the ComponentContext.getAttributeNames() method; in the next version, the use of this field is replaced by the usage of the java.lang.Collections.EMPTY_LIST.[15]

[15] To be clear, this is the *only* usage of EmptyIterator in STRUTS.

Secondly, the correct replacement functionality is outside of the STRUTS library: as AURA only considers viable replacements from within the STRUTS library, it would be correct for AURA to mark these methods as deleted in this context.

```
1  public class InsertTag
2    extends DefinitionTagSupport
3    implements PutTagParent, ComponentConstants, PutListTagParent {
4  /*Code elided*/
5    /**
6     * Get flush.
7     */
8    public boolean getFlush() {
9      return flush;
10   }
11 /*Code elided*/
12 }
13
14 public interface AttributeDefinition extends Serializable
15 {
16
17   /**
18    * Return value hold by this typed attribute.
19    */
20   public Object getValue();
21 /*Code elided*/
22 }
23
24 public class TilesPlugin implements PlugIn {
25 /*Code elided*/
26     /**
27      * End plugin.
28      */
29     public void destroy() {
30         definitionFactory.destroy();
31         definitionFactory = null;
32     }
33 /*Code elided*/
34 }
```

Listing D.12: The getAttributeNames() method from the ComponentContext class, v1.2.4

Listing D.12 presents the three replacement methods recommended by AURA, in their enclosing classes: the order of the replacements matches the order of the deleted methods from Listing D.9. It should be clear that the identified replacements bear *little* resemblance to the originals. The method names are entirely different, the inheritance hierarchy of their containing classes are different in each case, and one of the recommended replacements is a method in an interface, and

thus has no implementation. The only points in common between these recommendations, and the original functionality, are their return types.

Why did AURA *choose these replacements?* In their description of the AURA tool's implementation, Wu et al. note that in cases where there are multiple lexical matches that all appear to be equivalent, one is picked at random as the correct match [Wu et al., 2010, Section 5, p.329]. It is my speculation that this is how AURA has made these selections. A lexical comparison based on method signatures alone will indicate there is *some* similarity between the original methods, and these replacements; this similarity must have been high enough to pass whatever internal threshold Wu et al. set.

As these return types are common, it is likely that AURA would have found multiple method replacements for each that had the same lexical similarity; if it chose the replacement at random from these, it would explain why the replacements it identifies for these three methods have such varied contexts. These three changes should be reported as a mistake by the AURA tool. However, in my comparisons using AURA's results in Chapter 6 and Chapter 7, I have excluded these mistakes as they are changes to non-BI entities.

D.4.4 Changes on which the Corpus and Wu et al. Disagree

Table D.12 lists the API changes made to the STRUTS API that introduced binary incompatibilities, but for which the Corpus and the AURA tool disagree. I argue that in these cases that the recommended replacements (when made) are incorrect. Because of the number of changes on which the Corpus and AURA disagree in Table D.12, providing the detailed rationale for every case would take a considerable amount of space. Instead, I have chosen to provide a detailed examples of four changes that are representative of the mistakes AURA made in its analysis.

Table D.12: Wu et al. API Changes in Disagreement. (All in org.apache.struts)

Original	Replacement
action.ActionServletWrapper.log	mock.MockServletContext.log
config.ConfigHelper.errorOutput	config.ConfigHelper.getToken
config.ConfigHelper.errorOutput	config.ConfigHelper.message
config.ConfigHelper.errors	action.ActionMessages.properties
config.ConfigHelper.errors	config.ConfigHelper.action
config.ConfigHelper.errorsEmpty	config.ConfigHelper.ConfigHelper
config.ConfigHelper.errorSize	config.ConfigHelper.ConfigHelper
config.ConfigHelper.errorSize	config.ConfigHelper.isMessage
config.ConfigHelper.getActionErrors	config.ConfigHelper.getActionForm
config.ConfigHelper.getActionFormBeans	none
config.ConfigHelper.getActionForwards	none
config.ConfigHelper.getActionMappings	none
config.ConfigHelper.getErrorOutput	config.ConfigHelper.getLink
config.ConfigHelper.getErrorOutput	config.ConfigHelper.getOrigRef
config.ConfigHelper.getErrors	config.ConfigHelper.getException
config.ConfigHelper.getErrors	config.ConfigHelper.getMessage
config.ConfigHelper.getErrorsEmpty	config.ActionConfig.getUnknown
config.ConfigHelper.getErrorSize	config.ConfigHelper.getAction
config.ConfigHelper.getErrorSize	upload.FormFile.getFileSize
config.ConfigHelperInterface.getActionFormBeans	config.ConfigHelperInterface.getActionForm
config.ConfigHelperInterface.getActionForwards	config.ConfigHelperInterface.getActionMessages
config.ConfigHelperInterface.getActionMappings	config.ConfigHelperInterface.getActionMessages
config.ConfigHelperInterface.getErrorOutput	config.ConfigHelperInterface.getAction

Continued on next page

Table D.12 — continued from previous page

Original	Replacement
config.ConfigHelperInterface.getErrorOutput	config.ConfigHelperInterface.getToken
config.ConfigHelperInterface.getErrors	config.ConfigHelperInterface.getException
config.ConfigHelperInterface.getErrors	config.ConfigHelperInterface.getLink
config.ConfigHelperInterface.getErrorsEmpty	config.ConfigHelperInterface.getServletMapping
config.ConfigHelperInterface.getErrorSize	config.ConfigHelperInterface.isMessage
config.ConfigHelperInterface.getErrorSize	config.FormPropertyConfig.getSize
config.ConfigHelperInterface.getLocale	upload.MultipartRequestWrapper.getLocale
config.ControllerConfig.getDebug	config.ControllerConfig.getBufferSize
config.ControllerConfig.setDebug	config.ControllerConfig.setBufferSize
config.FormBeanConfig.getModuleConfig	config.FormBeanConfig.getDynaActionFormClass
taglib.html.FormTag.getScope	taglib.html.FormTag.getMethod
taglib.html.FormTag.getType	taglib.html.FormTag.getMethod
taglib.html.FormTag.setName	taglib.html.ImgTag.setName
taglib.html.FormTag.setScope	taglib.html.FormTag.setMethod
taglib.html.FormTag.setType	taglib.html.FormTag.setMethod

1. The org.apache.struts.action.ActionServletWrapper.log() method(s), for which AURA's identified replacements came from the test suite for the STRUTS API, rather than the STRUTS API.

2. The org.apache.struts.config.ConfigHelperInterface.getLocale() method, for which the identified replacement provides only partial functionality, and is inferior to the one identified in the Corpus.

3. The org.apache.struts.taglib.html.FormTag.getScope() and org.apache.struts.taglib.html.

FormTag.getType() methods which AURA identifies as having the same replacement. I will show the replacement is not equivalent, and the Corpus provides a better adaptation for both methods.

4. The org.apache.struts.config.ConfigHelper.getActionFormBeans() method, which AURA identifies as being deleted, but for which the Corpus identified a replacement within the STRUTS library.

D.4.4.1 **org.apache.struts.action.ActionServletWrapper.log()**.

```
1  public class ActionServletWrapper {
2      /**
3       * The controller servlet instance to which we are attached.
4       */
5      protected transient ActionServlet servlet = null;
6      /**
7       * Log the specified message if the current debugging detail level for
8       * this servlet has been set to an equal or higher value. Otherwise,
9       * ignore this message.
10      *
11      * @param message Message to be logged
12      * @param level Debugging detail level of this message
13      * @deprecated Logging should now use the commons logging
14      */
15     public void log(String message, int level) {
16
17             servlet.log(message,level);
18
19     }
20     /**
21      * Log message.
22      * @param message
23      */
24     public void log(String message) {
25
26              servlet.log(message);
27
28     }
29 //Code elided
30 }
```

Listing D.13: The ActionServletWrapper class, v1.1.0

Listing D.13 shows both of the log() methods contained in the org.apache.struts.action.ActionServlet-Wrapper class. One of the two methods is deprecated, and indicates that logging "...should now

```
1  /**
2   * <p>Mock <strong>ServletContext</strong> object for low-level unit tests
3   * of Struts controller components. Coarser grained tests should be
4   * implemented in terms of the Cactus framework, instead of the mock
5   * object classes.</p>
6   *
7   * <p><strong>WARNING</strong> - Only the minimal set of methods needed to
8   * create unit tests is provided, plus additional methods to configure this
9   * object as necessary. Methods for unsupported operations will throw
10  * <code>UnsupportedOperationException</code>.</p>
11  *
12  * <p><strong>WARNING</strong> - Because unit tests operate in a single
13  * threaded environment, no synchronization is performed.</p>
14  *
15  * @version $Revision: 1.5 $ $Date: 2004/03/14 06:23:52 $
16  */
17
18 public class MockServletContext implements ServletContext {
19 //Code elided
20     /**
21      * Default destination for <code>log()</code> output.
22      */
23     protected Log log = LogFactory.getLog(MockServletContext.class);
24 //Code elided
25     public void log(Exception exception, String message) {
26         log(message, exception);
27     }
28
29
30     public void log(String message) {
31         log.info(message);
32     }
33
34
35     public void log(String message, Throwable throwable) {
36         log.error(message, throwable);
37     }
38 //Code elided
39
40 }
```

Listing D.14: The MockServletContext class, v1.2.4

use the commons logging", which refers to the introduction of the APACHE COMMONS LOGGING (JCL) library.[16] JCL is a generic API facade for logging libraries that attempts to provide a cross-library, standard logging API that can be "wrapped" around any desired logging library, using a custom-written adaptor. The comment is indicating that logging functionality should be provided

[16] http://commons.apache.org/proper/commons-logging/, JCL is an acronym for the JAVA COMMONS LOGGING library.

through this external library, and no longer through the STRUTS API. In the next API version, both methods are removed.

AURA identifies the replacement for the ActionServletWrapper.log() method as being the org.apache.struts.mock.MockServletContext.log() method. However (as the name might hint), this is not a method in the STRUTS API; rather, it is a mock object used for testing, and is part of the test suite for STRUTS library. Listing D.14 shows the relevant code for MockServletContext, including the class comment that states it is intended for low-level unit testing of STRUTS components.

It should be superficially clear that AURA's recommendation is wrong in this case; however, I will provide additional concrete reasons to counter the argument that this could yet be a viable means for accessing logging functionality. First, the documented change in the source code clearly states that logging is moved to external functionality; while Chapter 5 does present cases where source-code documentation was wrong, it was only shown to be wrong in cases where functionality was referred to that did not exist. That is not the case here.

Secondly, the implementation of the MockServletContext.log() methods show that it relies on the org.apache.commons.logging.LogFactory class, from the JCL library, to provide logging functionality. However, for LogFactory to work, the underlying logging system (for which JCL acts as a facade) must be configured; in the absence of this configuration, JCL is designed to silently fail, but without causing an error, or throwing an exception.[17] Thus, a developer who mistakenly uses AURA's recommendation here will not understand why their attempts to adding logging information have no effect.

D.4.4.2 **org.apache.struts.config.ConfigHelperInterface.getLocale()**.

Listing D.15 shows the implementation and context of the getLocale() method in the org.apache.struts.config.ConfigHelperInterface. Because this is an interface, getLocale() does not have an implementation, making it difficult to understand what its replacement should be. To ground this example, Listing D.16 presents the implementation of getLocale() in the org.apache.struts.config.

[17] http://commons.apache.org/proper/commons-logging/guide.html

ConfigHelper class, which implements the ConfigHelperInterface interface. ConfigHelper is the only class in the STRUTS API which implements this interface, and so callers of the getLocale() method in the API are calling this particular implementation.[18]

```
1  public interface ConfigHelperInterface {
2  //Code elided
3      /**
4       * The <code>java.util.Locale</code> for the user, if any.
5       * If a default locale object is not in the user's session,
6       * the system default locale is returned.
7       * If used, the user locale is typically set during login
8       * processing under the key <code>Globals.LOCALE_KEY</code>.
9       */
10     public Locale getLocale();
11 //Code elided
12 }
```

Listing D.15: The ConfigHelperInterface interface, v1.1.0

The implementation of the method, as explained in its documentation, is in two parts. The method first attempts to return the locale of the user who is currently using the system, which hopefully has been established during their interaction with the STRUTS-based website (e.g., through explicitly setting their region, creating a user profile, etc.). To do so, it accesses the javax.servlet.http.HttpSession object that tracks a single user's information across the website.

If an HttpSession object is not available (either because the user is new, not logged in, or the implementation has decided to not use this feature), it instead accesses a default locale[19] stored in the javax.servlet.http.HttpServletRequest object, representing communication from the user to the website. Additional checks are made to ensure that a **null** pointer exception is not triggered during its attempt to resolve the locale, although a **null** pointer is returned should this fail to be resolved.

AURA identifies the org.apache.struts.upload.MultipartRequestWrapper.getLocale() method, shown in Listing D.17, as the replacement for ConfigHelperInterface.getLocale(). As with the earlier MockServletContext.log() recommendation, AURA has also recommended an accidentally similar method.

[18] ConfigHelper is not subclassed in the STRUTS library either.

[19] Details can be found in the JAVADOC documentation for javax.servlet.ServletRequest here: http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/api/index.html

```
1  public class ConfigHelper implements ConfigHelperInterface {
2  //Code elided
3      /**
4       * The session associated with this instance.
5       */
6      private HttpSession session = null;
7  //Code elided
8      /**
9       * The request associated with this instance.
10      */
11     private HttpServletRequest request = null;
12 //Code elided
13     /**
14      * The <code>java.util.Locale</code> for the user, if any.
15      * If a default locale object is not in the user's session,
16      * the system default locale is returned.
17      * If used, the user locale is typically set during login
18      * processing under the key <code>Globals.LOCALE_KEY</code>.
19      */
20     public Locale getLocale() {
21         Locale locale = null;
22
23         if (session != null) {
24             locale = (Locale) session.getAttribute(Globals.LOCALE_KEY);
25         }
26
27         if ((locale == null) && (request != null)) {
28             locale = request.getLocale();
29         }
30
31         return locale;
32     }
33 //Code elided
34 }
```

Listing D.16: The ConfigHelper class, v1.1.0

Whereas ConfigHelper and ConfigHelperInterface are intended to expose shared resources on a website for use by other functionality,[20] MultipartRequestWrapper is a wrapper around an HttpServletRequest object, intended to make it easier to work with file uploads to a website.[21] MultipartRequestWrapper delegates its getLocale() implementation to use the equivalent method on the HttpServletRequest object that it has been "wrapped around".

Aside from being intended for a very different purpose, AURA's recommended replacement

[20] http://struts.apache.org/release/1.3.x/apidocs/org/apache/struts/config/ConfigHelperInterface.html

[21] http://struts.apache.org/release/1.3.x/apidocs/index.html?org/apache/struts/upload/MultipartRequestWrapper.html

```
1  public class MultipartRequestWrapper implements HttpServletRequest {
2  //Code elided
3      /**
4       * The underlying HttpServletRequest
5       */
6      protected HttpServletRequest request;
7  //Code elided
8      public Locale getLocale() {
9          return request.getLocale();
10     }
11 //Code elided
12 }
```

Listing D.17: The MultipartRequestWrapper class, v1.2.4

in this case makes no attempt to obtain a user-specific locale, and will always default to obtaining whatever locale was set in the request object. This is not an equivalent replacement.

```
1  public class RequestUtils {
2  //Code elided
3      /**
4       * <p>Look up and return current user locale, based on the specified parameters.</p>
5       *
6       * @param request The request used to lookup the Locale
7       * @param locale Name of the session attribute for our user's Locale. If this is
8       * <code>null</code>, the default locale key is used for the lookup.
9       * @return current user locale
10      * @since Struts 1.2
11      */
12     public static Locale getUserLocale(HttpServletRequest request, String locale) {
13         Locale userLocale = null;
14         HttpSession session = request.getSession(false);
15         if (locale == null) {
16             locale = Globals.LOCALE_KEY;
17         }
18         // Only check session if sessions are enabled
19         if (session != null) {
20             userLocale = (Locale) session.getAttribute(locale);
21         }
22         if (userLocale == null) {
23             // Returns Locale based on Accept-Language header or the server default
24             userLocale = request.getLocale();
25         }
26         return userLocale;
27     }
28 //Code elided
29 }
```

Listing D.18: The RequestUtils.getUserLocale() method, v1.2.4

The Corpus recommends the org.apache.struts.util.RequestUtils.getUserLocale() method, shown in Listing D.18. This method is static, so it can be accessed from any calling context, and does not need to act as a wrapper. Further, it checks for a user-specific locale, and returns that if it exists. While it requires a HttpServletRequest and String object as parameters, the String parameter can be **null** (in which case it defaults to the same behaviour as ConfigHelper does in v1.1.0). AURA's recommendation also requires access to a HttpServletRequest object, in order to instantiate an instance of a MultipartRequestWrapper object.

The other way in which this replacement is superior is shown in how the HttpSession object is accessed from the HttpServletRequest parameter in line 14; in providing the **false** parameter, the code instructs the HttpServletRequest object to not create a new HttpSession object if one does not already exist. In this way, it provides equivalent functionality to what ConfigHelper previously provided.

While MultipartRequestWrapper has access to the same parameters at its initialization, it does not perform this step. Thus, the RequestUtils.getUserLocale() method identified by the Corpus is a better replacement for the functionality lost by the removal of the ConfigHelperInteferace.getLocale() method. By comparison, AURA's recommended replacement provides only partial functionality, as it does not attempt to return the user's locale, but always returns whatever locale is associated with the servlet request.

D.4.4.3 **org.apache.struts.taglib.html.FormTag.getScope() and FormTag.getType()**.

Listing D.19 shows an excerpt from the org.apache.struts.taglib.html.FormTag class, containing the getScope() and getType() methods, as well as the fields they act as getters for. I have included two additional fields, beanScope and beanType, that replace the scope and type fields respectively in the next version, and whose code comment explains their purpose.

The FormTag class supports the generation of HTML code for creating forms (e.g., containing text fields, radio buttons, etc.) that capture user input; the fields and getters associated with this class allow accessing attribute information relevant to these items. In this case, the scope field

```
1  public class FormTag extends TagSupport {
2  //Code elided
3      /**
4       * The scope of the form bean to (create and) use. This is either the same
5       * as the 'scope' attribute, if that was specified, or is obtained from the
6       * associated <code>ActionMapping</code> otherwise.
7       */
8      protected String beanScope = null;
9      /**
10      * The type of the form bean to (create and) use. This is either the same
11      * as the 'type' attribute, if that was specified, or is obtained from the
12      * associated <code>ActionMapping</code> otherwise.
13      */
14     protected String beanType = null;
15 //Code elided
16     /**
17      * The scope (request or session) under which our associated bean
18      * is stored.
19      */
20     protected String scope = null;
21 //Code elided
22     /**
23      * The Java class name of the bean to be created, if necessary.
24      */
25     protected String type = null;
26 //Code elided
27     /**
28      * Return the attribute scope of our bean.
29      */
30     public String getScope() {
31         return (this.scope);
32     }
33 //Code elided
34     /**
35      * Return the Java class of our bean.
36      */
37     public String getType() {
38         return (this.type);
39     }
40 //Code elided
41 }
```

Listing D.19: The getScope() and getType() methods in the FormTag class, v1.1.0

and getScope() method are referring to an attribute indicating the scope of this particular form, which the code documentation shows is meant to reflect that it is tied to an http *request* or *session*. Similarly, the type field and getType() method are referring to the qualified name of the JAVA class that has been used to generate this form; each different kind of form element will likely have its

own specialized subclass.

```
1  public class FormTag extends TagSupport {
2  //Code elided
3      /**
4       * The request method used when submitting this form.
5       */
6      protected String method = null;
7  //Code elided
8      /**
9       * Return the request method used when submitting this form.
10      */
11     public String getMethod() {
12
13         return (this.method);
14
15     }
16 //Code elided
17 }
```

Listing D.20: The getMethod() method from the FormTag class, v1.2.4

Listing D.20 shows the replacement AURA identifies for the getScope() and getType() methods, FormTag.getMethod(); this method was present in the v1.1.0 of the STRUTS API as well. The getMethod() however is intended to indicate how a form is being submitted (i.e. via an http GET or POST request), rather than indicating the type of the object generating this form, or whether its data is part of an http request or session.[22] Clearly, this information is incompatible with what the previous two methods expect: should a developer use this instead, they would receive nonsensical information that does not match what they have previously received from these methods.

Instead, the Corpus recommends the usage of the javax.servlet.jsp.tagext.TagSupport.getValue() method. TagSupport is the parent class of FormTag, and stores attribute data about itself, and implementing subclasses, in an attribute-Object mapping. To explain how this acts as a replacement, I provide an example in Listing D.21 of a method which relied on the getScope() method in the STRUTS 1.1.0 API. Migrating this usage to the next library version requires that: (i) the method used is changed to getValue();[23] (ii) the attribute name "scope" is provided; and, (iii) the returned value is cast to the String type. The migrated code is shown in Listing D.22. Note that no other

[22] http://struts.apache.org/development/1.x/struts-taglib/tlddoc/index.html

[23] Because TagSupport is the parent class of FormTag, the method can be invoked on FormTag objects.

changes needed to be made to code for the migration; only one line was changed.

```
1  public void checkFormScope( FormTag current_tag ) {
2
3    String form_scope = current_tag.getScope();
4    if ( form_scope.equals( "request" ) ) {
5    //Code elided
6    }
7    else if ( form_scope.equals( "session" ) {
8    //Code elided
9    }
10   else {
11     LOGGER.error( "Unknown scope in FormTag" );
12   }
13 }
```

Listing D.21: Example method using the getScope() method from the FormTag class, v1.1.0

```
1  public void checkFormScope( FormTag current_tag ) {
2
3    String form_scope = ( String )current_tag.getValue( "scope" );
4    if ( form_scope.equals( "request" ) ) {
5    //Code elided
6    }
7    else if ( form_scope.equals( "session" ) {
8    //Code elided
9    }
10   else {
11     LOGGER.error( "Unknown scope in FormTag" );
12   }
13 }
```

Listing D.22: Example method using the getValue() method from the TagSupport class, v1.2.4

Similarly, migrating code using the FormTag.getType method would use the same method (TagSupport.getValue()), but supply "type" as the attribute parameter.

D.4.4.4 **org.apache.struts.config.ConfigHelper.getActionFormBeans()**.

Listing D.23 presents the implementation of the org.apache.struts.config.ConfigHelper.getActionFormBeans() method in the ConfigHelper class. This is one of several of methods that AURA has marked as being deleted without replacement from the STRUTS API. Earlier in this section, I noted that this was correct for cases where the replacement noted in the Corpus was external to the STRUTS API, as Wu et al. do not consider external replacements. However, the Corpus indicates a replacement

within the STRUTS API is available: discovering this required a detailed analysis of the library's implementation, which I will guide the reader through.

```
1  public class ConfigHelper implements ConfigHelperInterface {
2  //Code elided
3      /**
4       * The application associated with this instance.
5       */
6      private ServletContext application = null;
7  //Code elided
8      /**
9       * The <code>org.apache.struts.action.ActionFormBeans</code> collection
10      * for this application.
11      */
12     public ActionFormBeans getActionFormBeans() {
13         if (this.application == null) {
14             return null;
15         }
16         return (ActionFormBeans) this.application.getAttribute(Globals.FORM_BEANS_KEY);
17     }
18 //Code elided
19 }
```

Listing D.23: Example method using the getScope() method from the FormTag class, v1.1.0

While it is not visible in Listing D.23, a modern IDE will mark the ActionFormBeans return type with a strike-out (i.e., ~~ActionFormBeans~~) to notify the developer that ActionFormBeans is a deprecated type. The implementation of ActionFormBeans, along with much of its code comments, is provided in Listing D.24. The ActionFormBeans class was effectively a wrapper for housing a collection of ActionFormBean classes,[24] mapping them to their names for retrieval. The implementation of ConfigHelper.getActionFormBeans() indicates that this collection was being stored in the website's ServletContext, itself containing an attribute-Object map, in which this data was stored using the Globals.FORM_BEANS_KEY identifier. This suggests that the collection of ActionFormBeans was not specific to one particular session or user, and could be accessed globally as needed.

A clue about the replacement for the getActionFormBeans() method can be found by examining the code commends contained in the ActionFormBeans class. The @deprecated tag in the class's

[24] ActionFormBeans and ActionFormBean are two different classes, and I have been careful to ensure I am referring to each appropriately in this writeup. I will not comment on the design decision that distinguished two related classes by a single letter at the end of their names.

comment indicates the entire class was being replaced with FormBeanConfig objects, which can be accessed through the org.apache.struts.config.ModuleConfig interface.[25] The ActionFormBean class is a subclass of FormBeanConfig; thus if the ActionFormBeans class is being replaced by methods which return a set of FormBeanConfig objects, it is possible that those methods will provide equivalent functionality to ConfigHelper.getActionFormBeans(). Because ModuleConfig is an interface though, it would be helpful to examine a concrete implementation.

Listing D.25 shows the implementation of methods related to the FormBeanConfig type in the ModuleConfigImpl class, a concrete reification of the ModuleConfig interface.[26] Since the ConfigHelper .getActionFormBeans() method returned an ActionFormBeans object, which was effectively a wrapper over a hash map, something which returns a hash map of FormBeanConfig objects could be a viable replacement. Unfortunately, none do; however, the ModuleConfigImpl.findFormBeanConfigs() method returns an array of FormBeanConfig objects. Effectively, this would be the set of objects in the hash map, but without the mapping index.

However, if we refer back to the original implementation of the ActionFormBeans class, we notice that according to the addFormBean() method, the hash map containing the ActionFormBean objects is indexed by their "bean name", provided through ActionFormBean.getName(). Thus, it is possible for us to either: (a) construct a hash map that indexes these objects according to their name; or, (b) iterates over the array to find the object whose name matches what we are looking for. This would appear to provide us with equivalent functionality.

The Corpus has marked the replacement of ConfigHelper.getActionFormBeans() as this ModuleConfig.findFormBeanConfigs() method. Note however that in the "Transformation" column, the change has been marked as "1:n": because some additional adaptive work must be done to incorporate this replacement: it is not a "1:1" replacement. To see why, consider a portion of code which relied on the original implementation of ConfigHelper.getActionFormBeans(), shown in Listing D.26.

[25] It is worth noting here that: (1) the original ConfigHelper.getActionFormBeans() method was not deprecated prior to its removal; (2) no replacement was specified for the method; and, (3) even in this case, the code comment for ActionFormBeans does not specify which methods in this class are replaced by which methods in ModuleConfig.

[26] The code has been preserved in the order it occurs in its original implementation.

As part of this method's implementation, a call is made to ConfigHelper.getActionFormBeans() retrieve a pointer to the stored hash map of ActionFormBeans. In this case, the developer wants to retrieve two ActionFormBean objects that are associated with login and password forms on a webpage; having retrieved the wrapped hash map, she then uses the ActionFormBeans.findFormBean() method to retrieve those specific objects, by using a unique identifier previously specified for those form fields (e.g., "login.userid", etc.). The rest of this method's implementation is elided.

To migrate to the new STRUTS 1.2.4 version, the developer now uses the ModuleConfig.findFormBeanConfigs() method, as shown in Listing D.27. But, she must rework an array of FormBeanConfig objects into a hash map of ActionFormBean objects if she is to incorporate this into her code.[27] To do so, she first creates her own HashMap object. Then, using the knowledge that these objects were indexed by the ActionFormBean.getName() method, she constructs her hash map by iterating over the array, casting each FormBeanConfig object to ActionFormBeanConfig, and storing it by its name in the hash map. Now she can use this hash map in place of the former ActionFormBeans object, using the HashMap.get() method instead of the previous findFormBean() method.

D.4.5 Comparison with Wu et al.: Summary

Table D.13: Comparison of AURA to the Corpus for STRUTS 1.1.0 to 1.2.4.

Changes Identified	BI Changes		Non-BI Changes		Non-STRUTS Changes	
	Agrees	Disagrees	Correct	Error	Test Suite	External
135	34	38	10	3	26	24

Table D.13 summarizes the comparison of the AURA tool's identified change rules, to those of the Corpus for the STRUTS v1.1.0 to v1.2.4 migration. Of the 135 changes AURA identifies, 72 concern binary incompatible changes, of which 34 are in agreement with the Corpus. Based on the evidence present in the source code of each STRUTS version (see Section D.4.4), I found the other 38 change rules AURA identifies for binary incompatible changes to be errors.

[27] Astute readers will note that there are more efficient means of enacting this migration; however, my choice here to modify as little of the surrounding code as possible reflects the desire to illustrate how this one method is replaced.

The other change rules AURA identifies comes mainly from code external to the STRUTS API implementation (50 in total), although 13 changes were identified to non-BI code in the STRUTS library. Ten of these changes are either backwards-compatible API changes, or internal modifications that do not affect the API or alter the library's behaviour. Three changes were internal modifications where AURA's identified replacement was erroneous.

There are a number of cases where AURA's analysis is mistaken, most notably in recommending replacements for binary incompatible changes. Broadly speaking, these mistakes seem to occur in cases where call analysis yielded no useful matches, but there were other entities in the new API version that had some lexical similarity to those that were removed. Their lexical similarity seemed to be high enough that AURA recommended incorrect functionality as a replacement, sometimes in cases where there were other signs that these should not be matches.

D.5 Comparison with Meng et al.

Meng et al. have made the evaluation of their HIMA tool on the STRUTS v1.1.0 to v1.2.4 migration available online.[28] Like Schäfer et al. [2008] and Wu et al. [2010], HIMA reports change rules it detects as having occurred in the library's API between versions, but also indicates whether the authors have judged that change rule as being correct. In addition, Meng et al. also report those methods they believe were deleted without replacement from the API, and also indicate whether their investigation found that this was correct. I compare both their detected change rules, and detected deletions, against the API changes recorded in the Corpus. The data presented in the following tables is as recorded in the provided data for the "Struts & Task 1" evaluation of HIMA, which is the paper's label for the STRUTS v1.1.0 to v1.2.4 migration.

Meng et al. report HIMA as detecting 108 one-to-one change rules for the STRUTS v1.1.0 to v1.2.4 migration, of which the paper claims 107 are correct; similarly, 76 instances of functionality

[28] http://sourceforge.net/projects/hima/files/ExperimentResult. Note that two sets of data are available; having analyzed both, it appears that the "hima_results_all" dataset matches what is reported in the paper. The number of change rules identified in "HiMA_results_new" is dramatically different. The comparisons in this section are taken from the "hima_results_all" dataset.

being simply deleted are recorded, with the paper indicating that all 76 are correct. Their paper reports HIMA as having 99.1% precision in detecting one-to-one change rules, and 100% precision in detecting deleted functionality. Recall is reported qualitatively: the paper states that HiMA's recall is better than that of the AURA tool in the libraries examined [Meng et al., 2012, pp. 360, 362]. Only one-to-one, and deletion rules are detected: the paper reports no one-to-many, or many-to-one changes detected in STRUTS. Their provided datafile differs slightly from the above accounting. Their data shows 107 change rules are detected by HIMA, all marked by the authors as correct; the erroneous change rule is not indicated, or not present in that data. Similarly, their data reports 77 "deletions" of methods, with 76 marked as correct, and one marked as wrong. It is possible that one of these results was misclassified as a change rule in the original paper.

Unfortunately, I was not aware of this paper and the HIMA tool until late in the writing process of this work. While I am able to present a detailed comparison of the HIMA tool's detected change rules for the STRUTS v1.1.0 to 1.2.4 migration, due to time constraints I was not able to prepare for the dissertation the detailed examples demonstrating why HIMA's disagreements with the Corpus are due to errors it makes, as I was able to for earlier sections.

D.5.1 Changes on which Meng et al. and the Corpus Agree

Table D.14: Meng et al. Matched API Changes. (All in org.apache.struts)

Original	Replacement
action.ActionMapping.findException(java.lang.Class)	config.ActionConfig.findException(java.lang.Class)
action.ActionMessages.ActionMessageItem. ActionMessageItem(java.util.List, **int**)	action.ActionMessages.ActionMessageItem. ActionMessageItem(java.util.List, **int**, java.lang.String)
config.ConfigHelper.getLocale()	util.RequestUtils.getUserLocale(HttpServletRequest request, String locale)
taglib.html.FrameTag.getMarginheight()	taglib.html.FrameTag.getMarginheight()

Continued on next page

Table D.14 — continued from previous page

Original	Replacement
taglib.html.FrameTag.getMarginwidth()	taglib.html.FrameTag.getMarginwidth()
taglib.html.FrameTag.setMarginheight(**int**)	taglib.html.FrameTag.setMarginheight(java.lang.Integer)
taglib.html.FrameTag.setMarginwidth(**int**)	taglib.html.FrameTag.setMarginwidth(java.lang.Integer)
tiles.UntyppedAttribute.getRole()	tiles.UntypedAttribute.getRole()
tiles.UntyppedAttribute.getValue()	tiles.UntypedAttribute.getValue()
tiles.UntyppedAttribute.setRole(java.lang.String)	tiles.UntypedAttribute.setRole(java.lang.String)
tiles.UntyppedAttribute.setValue(java.lang.Object)	tiles.UntypedAttribute.setValue(java.lang.Object)
tiles.UntyppedAttribute.toString()	tiles.UntypedAttribute.toString()
tiles.UntyppedAttribute.UntyppedAttribute(java.lang.Object)	tiles.UntypedAttribute.UntypedAttribute(java.lang.Object)
tiles.UntyppedAttribute.UntyppedAttribute(java.lang.Object, java.lang.String)	tiles.UntypedAttribute.UntypedAttribute(java.lang.Object, java.lang.String)
validator.FieldChecks.validateByte(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateByte(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateCreditCard(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateCreditCard(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateDate(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateDate(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateDouble(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateDouble(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)

Continued on next page

Table D.14 — continued from previous page

Original	Replacement
validator.FieldChecks.validateDoubleRange(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateDoubleRange(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateEmail(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateEmail(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateFloat(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateFloat(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateFloatRange(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateFloatRange(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateInteger(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateInteger(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateIntRange(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateIntRange(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateLong(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateLong(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)

Continued on next page

Table D.14 — continued from previous page

Original	Replacement
validator.FieldChecks.validateMask(java.lang.Object, validator. ValidatorAction, validator.Field, validator.ActionErrors, validator .HttpServletRequest)	validator.FieldChecks.validateMask(java.lang.Object, validator. ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateMaxLength(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateMaxLength(java.lang.Object, validator.ValidatorAction, validator.Field, validator. ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateMinLength(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateMinLength(java.lang.Object, validator.ValidatorAction, validator.Field, validator. ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateRequired(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, validator.HttpServletRequest)	validator.FieldChecks.validateRequired(java.lang.Object, validator.ValidatorAction, validator.Field, validator. ActionMessages, validator.HttpServletRequest)
validator.FieldChecks.validateRequiredIf(java.lang.Object, validator.ValidatorAction, validator.Field, validator.ActionErrors, org.apache.commons.validator.Validator, validator. HttpServletRequest)	validator.FieldChecks.validateRequiredIf(java.lang.Object, validator.ValidatorAction, validator.Field, validator. ActionMessages, org.apache.commons.validator.Validator, validator.HttpServletRequest)
validator.FieldChecks.validateShort(java.lang.Object, validator. ValidatorAction, validator.Field, validator.ActionErrors, validator .HttpServletRequest)	validator.FieldChecks.validateShort(java.lang.Object, validator. ValidatorAction, validator.Field, validator.ActionMessages, validator.HttpServletRequest)
validator.Resources.initValidator(java.lang.String, java.lang. Object, validator.ServletContext, validator.HttpServletRequest, validator.ActionErrors, **int**)	validator.Resources.initValidator(java.lang.String, java.lang. Object, validator.ServletContext, validator.HttpServletRequest, validator.ActionMessages, **int**)

Table D.14 lists the binary incompatible API changes detected by HIMA for which HIMA's recommended replacement is in agreement with that of the Corpus. A total of 32 BI methods have their adaptations correctly identified by HIMA.

Table D.15: Meng et al. Deleted API methods for which replacement functionality is external to STRUTS.

Original
share.org.apache.struts.action.ActionServletWrapper.log(java.lang.String)
share.org.apache.struts.config.ControllerConfig.getDebug()
share.org.apache.struts.config.ControllerConfig.setDebug(**int**)

Table D.15 lists binary incompatible API changes detected by HIMA that were reported as "simply-deleted", meaning HIMA had determined that no replacement functionality existed in the library. For the changes in this table, the Corpus identified them as BI changes, and identifies replacement functionality; however, the Corpus's identified replacement is external to the STRUTS library. All three of these methods represent logging functionality that previously was provided in the STRUTS library; as discussed in Section D.4.4, logging functionality was moved into an external library (the APACHE COMMONS JCL library) which was now expected to be incorporated. Because the replacements are external to STRUTS, HIMA's determination that these methods were deleted is not unreasonable given that it only analyzes the STRUTS library.

D.5.2 Non-Library Changes Considered by Meng et al.

Table D.16: Meng et al. Matched API Changes. (All in org.apache.struts)

Original	Replacement
webapp.example.LogonAction.getUser(webapp.example. UserDatabase, java.lang.String)	webapp.example.LogonAction.getUser(webapp.example. UserDatabase, java.lang.String, java.lang.String, webapp. example.ActionMessages)

Continued on next page

Table D.16 — continued from previous page

Original	Replacement
webapp.upload.UploadAction.execute(webapp.upload. ActionMapping, webapp.upload.ActionForm, webapp.upload. HttpServletRequest, webapp.upload.HttpServletResponse)	examples.org.apache.struts.webapp.upload.UploadAction. execute(examples.org.apache.struts.webapp.upload. ActionMapping, examples.org.apache.struts.webapp.upload. ActionForm, examples.org.apache.struts.webapp.upload. HttpServletRequest, examples.org.apache.struts.webapp. upload.HttpServletResponse)
webapp.upload.UploadForm.getFilePath()	examples.org.apache.struts.webapp.upload.UploadForm. getFilePath()
webapp.upload.UploadForm.getQueryParam()	examples.org.apache.struts.webapp.upload.UploadForm. getQueryParam()
webapp.upload.UploadForm.getTheFile()	examples.org.apache.struts.webapp.upload.UploadForm. getTheFile()
webapp.upload.UploadForm.getTheText()	examples.org.apache.struts.webapp.upload.UploadForm. getTheText()
webapp.upload.UploadForm.getWriteFile()	examples.org.apache.struts.webapp.upload.UploadForm. getWriteFile()
webapp.upload.UploadForm.reset()	examples.org.apache.struts.webapp.upload.UploadForm.reset ()
webapp.upload.UploadForm.setFilePath(java.lang.String)	examples.org.apache.struts.webapp.upload.UploadForm. setFilePath(java.lang.String)
webapp.upload.UploadForm.setQueryParam(java.lang.String)	examples.org.apache.struts.webapp.upload.UploadForm. setQueryParam(java.lang.String)

Continued on next page

Table D.16 — continued from previous page

Original	Replacement
webapp.upload.UploadForm.setTheFile(webapp.upload. FormFile)	examples.org.apache.struts.webapp.upload.UploadForm. setTheFile(examples.org.apache.struts.webapp.upload. FormFile)
webapp.upload.UploadForm.setTheText(java.lang.String)	examples.org.apache.struts.webapp.upload.UploadForm. setTheText(java.lang.String)
webapp.upload.UploadForm.setWriteFile(**boolean**)	examples.org.apache.struts.webapp.upload.UploadForm. setWriteFile(**boolean**)
webapp.upload.UploadForm.validate(webapp.upload. ActionMapping, webapp.upload.HttpServletRequest)	examples.org.apache.struts.webapp.upload.UploadForm. validate(examples.org.apache.struts.webapp.upload. ActionMapping, examples.org.apache.struts.webapp.upload. HttpServletRequest)
validator.org.apache.struts.webapp.validator.CityStateZip. getCity()	examples.org.apache.struts.webapp.validator.CityStateZip. getCity()
validator.org.apache.struts.webapp.validator.CityStateZip. getStateProv()	examples.org.apache.struts.webapp.validator.CityStateZip. getStateProv()
validator.org.apache.struts.webapp.validator.CityStateZip. getZipPostal(**int**)	examples.org.apache.struts.webapp.validator.CityStateZip. getZipPostal(**int**)
validator.org.apache.struts.webapp.validator.CityStateZip. setCity(java.lang.String)	examples.org.apache.struts.webapp.validator.CityStateZip. setCity(java.lang.String)
validator.org.apache.struts.webapp.validator.CityStateZip. setStateProv(java.lang.String)	examples.org.apache.struts.webapp.validator.CityStateZip. setStateProv(java.lang.String)
validator.org.apache.struts.webapp.validator.CityStateZip. setZipPostal(**int**, java.lang.String)	examples.org.apache.struts.webapp.validator.CityStateZip. setZipPostal(**int**, java.lang.String)

Continued on next page

Table D.16 — continued from previous page

Original	Replacement
validator.org.apache.struts.webapp.validator.LocaleAction. execute(validator.org.apache.struts.webapp.validator. ActionMapping, validator.org.apache.struts.webapp.validator. ActionForm, validator.org.apache.struts.webapp.validator. HttpServletRequest, validator.org.apache.struts.webapp. validator.HttpServletResponse)	examples.org.apache.struts.webapp.validator.LocaleAction. execute(examples.org.apache.struts.webapp.validator. ActionMapping, examples.org.apache.struts.webapp.validator. ActionForm, examples.org.apache.struts.webapp.validator. HttpServletRequest, examples.org.apache.struts.webapp. validator.HttpServletResponse)
validator.org.apache.struts.webapp.validator. MultiRegistrationAction.execute(validator.org.apache.struts. webapp.validator.ActionMapping, validator.org.apache.struts. webapp.validator.ActionForm, validator.org.apache.struts. webapp.validator.HttpServletRequest, validator.org.apache. struts.webapp.validator.HttpServletResponse)	examples.org.apache.struts.webapp.validator. MultiRegistrationAction.execute(examples.org.apache.struts. webapp.validator.ActionMapping, examples.org.apache.struts. webapp.validator.ActionForm, examples.org.apache.struts. webapp.validator.HttpServletRequest, examples.org.apache. struts.webapp.validator.HttpServletResponse)
validator.org.apache.struts.webapp.validator. MultiRegistrationAction.removeFormBean(validator.org.apache .struts.webapp.validator.ActionMapping, validator.org.apache. struts.webapp.validator.HttpServletRequest)	examples.org.apache.struts.webapp.validator. MultiRegistrationAction.removeFormBean(examples.org. apache.struts.webapp.validator.ActionMapping, examples.org. apache.struts.webapp.validator.HttpServletRequest)
validator.org.apache.struts.webapp.validator. RegistrationAction.execute(validator.org.apache.struts.webapp .validator.ActionMapping, validator.org.apache.struts.webapp. validator.ActionForm, validator.org.apache.struts.webapp. validator.HttpServletRequest, validator.org.apache.struts. webapp.validator.HttpServletResponse)	examples.org.apache.struts.webapp.validator. RegistrationAction.execute(examples.org.apache.struts. webapp.validator.ActionMapping, examples.org.apache.struts. webapp.validator.ActionForm, examples.org.apache.struts. webapp.validator.HttpServletRequest, examples.org.apache. struts.webapp.validator.HttpServletResponse)

Continued on next page

Table D.16 — continued from previous page

Original	Replacement
validator.org.apache.struts.webapp.validator.	examples.org.apache.struts.webapp.validator.
RegistrationAction.removeFormBean(validator.org.apache.	RegistrationAction.removeFormBean(examples.org.apache.
struts.webapp.validator.ActionMapping, validator.org.apache.	struts.webapp.validator.ActionMapping, examples.org.apache.
struts.webapp.validator.HttpServletRequest)	struts.webapp.validator.HttpServletRequest)
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
getAction()	RegistrationForm.getAction()
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
getAddr()	RegistrationForm.getAddr()
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
getCityStateZip()	RegistrationForm.getCityStateZip()
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
getEmail()	RegistrationForm.getEmail()
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
getFirstName()	RegistrationForm.getFirstName()
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
getLastName()	RegistrationForm.getLastName()
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
getPhone()	RegistrationForm.getPhone()
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
reset(validator.org.apache.struts.webapp.validator.	RegistrationForm.reset(examples.org.apache.struts.webapp.
ActionMapping, validator.org.apache.struts.webapp.validator.	validator.ActionMapping, examples.org.apache.struts.webapp.
HttpServletRequest)	validator.HttpServletRequest)

Continued on next page

Table D.16 — continued from previous page

Original	Replacement
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
setAction(java.lang.String)	RegistrationForm.setAction(java.lang.String)
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
setAddr(java.lang.String)	RegistrationForm.setAddr(java.lang.String)
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
setCityStateZip(validator.org.apache.struts.webapp.validator.	RegistrationForm.setCityStateZip(examples.org.apache.struts.
CityStateZip)	webapp.validator.CityStateZip)
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
setEmail(java.lang.String)	RegistrationForm.setEmail(java.lang.String)
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
setFirstName(java.lang.String)	RegistrationForm.setFirstName(java.lang.String)
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
setLastName(java.lang.String)	RegistrationForm.setLastName(java.lang.String)
validator.org.apache.struts.webapp.validator.RegistrationForm.	examples.org.apache.struts.webapp.validator.
setPhone(java.lang.String)	RegistrationForm.setPhone(java.lang.String)
validator.org.apache.struts.webapp.validator.TypeAction.	examples.org.apache.struts.webapp.validator.TypeAction.
execute(validator.org.apache.struts.webapp.validator.	execute(examples.org.apache.struts.webapp.validator.
ActionMapping, validator.org.apache.struts.webapp.validator.	ActionMapping, examples.org.apache.struts.webapp.validator.
ActionForm, validator.org.apache.struts.webapp.validator.	ActionForm, examples.org.apache.struts.webapp.validator.
HttpServletRequest, validator.org.apache.struts.webapp.	HttpServletRequest, examples.org.apache.struts.webapp.
validator.HttpServletResponse)	validator.HttpServletResponse)

Continued on next page

Table D.16 — continued from previous page

Original	Replacement
validator.org.apache.struts.webapp.validator.TypeAction. removeFormBean(validator.org.apache.struts.webapp.validator .ActionMapping, validator.org.apache.struts.webapp.validator. HttpServletRequest)	examples.org.apache.struts.webapp.validator.TypeAction. removeFormBean(examples.org.apache.struts.webapp. validator.ActionMapping, examples.org.apache.struts.webapp. validator.HttpServletRequest)
validator.org.apache.struts.webapp.validator.TypeForm. getAction()	examples.org.apache.struts.webapp.validator.TypeForm. getAction()
validator.org.apache.struts.webapp.validator.TypeForm. getByte()	examples.org.apache.struts.webapp.validator.TypeForm. getByte()
validator.org.apache.struts.webapp.validator.TypeForm. getCreditCard()	examples.org.apache.struts.webapp.validator.TypeForm. getCreditCard()
validator.org.apache.struts.webapp.validator.TypeForm. getDate()	examples.org.apache.struts.webapp.validator.TypeForm. getDate()
validator.org.apache.struts.webapp.validator.TypeForm. getDouble()	examples.org.apache.struts.webapp.validator.TypeForm. getDouble()
validator.org.apache.struts.webapp.validator.TypeForm. getFloat()	examples.org.apache.struts.webapp.validator.TypeForm. getFloat()
validator.org.apache.struts.webapp.validator.TypeForm. getFloatRange()	examples.org.apache.struts.webapp.validator.TypeForm. getFloatRange()
validator.org.apache.struts.webapp.validator.TypeForm. getInteger()	examples.org.apache.struts.webapp.validator.TypeForm. getInteger()
validator.org.apache.struts.webapp.validator.TypeForm. getLong()	examples.org.apache.struts.webapp.validator.TypeForm. getLong()

Continued on next page

Table D.16 — continued from previous page

Original	Replacement
validator.org.apache.struts.webapp.validator.TypeForm. getNameList()	examples.org.apache.struts.webapp.validator.TypeForm. getNameList()
validator.org.apache.struts.webapp.validator.TypeForm. getShort()	examples.org.apache.struts.webapp.validator.TypeForm. getShort()
validator.org.apache.struts.webapp.validator.TypeForm. initNames()	examples.org.apache.struts.webapp.validator.TypeForm. initNames()
validator.org.apache.struts.webapp.validator.TypeForm.reset(validator.org.apache.struts.webapp.validator.ActionMapping, validator.org.apache.struts.webapp.validator. HttpServletRequest)	examples.org.apache.struts.webapp.validator.TypeForm.reset(examples.org.apache.struts.webapp.validator.ActionMapping, examples.org.apache.struts.webapp.validator. HttpServletRequest)
validator.org.apache.struts.webapp.validator.TypeForm. setAction(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setAction(java.lang.String)
validator.org.apache.struts.webapp.validator.TypeForm. setByte(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setByte(java.lang.String)
validator.org.apache.struts.webapp.validator.TypeForm. setCreditCard(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setCreditCard(java.lang.String)
validator.org.apache.struts.webapp.validator.TypeForm. setDate(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setDate(java.lang.String)
validator.org.apache.struts.webapp.validator.TypeForm. setDouble(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setDouble(java.lang.String)
validator.org.apache.struts.webapp.validator.TypeForm. setFloat(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setFloat(java.lang.String)

Continued on next page

Table D.16 — continued from previous page

Original	Replacement
validator.org.apache.struts.webapp.validator.TypeForm. setFloatRange(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setFloatRange(java.lang.String)
validator.org.apache.struts.webapp.validator.TypeForm. setInteger(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setInteger(java.lang.String)
validator.org.apache.struts.webapp.validator.TypeForm. setLong(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setLong(java.lang.String)
validator.org.apache.struts.webapp.validator.TypeForm. setNameList(java.util.List)	examples.org.apache.struts.webapp.validator.TypeForm. setNameList(java.util.List)
validator.org.apache.struts.webapp.validator.TypeForm. setShort(java.lang.String)	examples.org.apache.struts.webapp.validator.TypeForm. setShort(java.lang.String)

Table D.17: Meng et al. Deleted methods from STRUTS examples. (All in org.apache.struts-.webapp.example)

Original
CheckLogonTag.getName()
CheckLogonTag.getPage()
CheckLogonTag.setName(java.lang.String)
CheckLogonTag.setPage(java.lang.String)
memory.MemoryDatabasePlugIn.setupCache(ActionServlet, ModuleConfig)
memory.MemorySubscriptionCreationFactory.– MemorySubscriptionCreationFactory(MemoryUserDatabase)

Table D.16 lists changes HIMA detected to code in example applications that were provided with the STRUTS library (see also Section D.4.2 for the same). Similarly, Table D.17 lists methods that HIMA detected as being deleted from these example applications without replacement, and Table D.18 lists methods detected as being deleted from the STRUTS test suite.

Because such changes are not part of the STRUTS library (or more specifically its API, in the

Table D.18: Meng et al. Deleted methods from the STRUTS test suite (All in org.apache.struts-.taglib).

Original
bean.TestCookieTag.beginCookieTagName(bean.WebRequest)
bean.TestCookieTag.beginCookieTagNameMultiple(bean.WebRequest)
bean.TestCookieTag.beginCookieTagNameValue(bean.WebRequest)
bean.TestCookieTag.endCookieTagName(bean.WebResponse)
bean.TestCookieTag.endCookieTagNameMultiple(bean.WebResponse)
bean.TestCookieTag.endCookieTagNameValue(bean.WebResponse)
bean.TestCookieTag.testCookieTagName()
bean.TestCookieTag.testCookieTagNameMultiple()
bean.TestCookieTag.testCookieTagNameValue()
bean.TestIncludeTag.endIncludeTagForward(bean.WebResponse)
bean.TestIncludeTag.endIncludeTagHref(bean.WebResponse)
bean.TestIncludeTag.endIncludeTagPage(bean.WebResponse)
bean.TestIncludeTag.formatAndTest(java.lang.String, java.lang.String)
logic.TestEqualTag.beginCookieStringEquals(logic.WebRequest)
logic.TestEqualTag.testCookieStringEquals()
logic.TestGreaterEqualTag.beginCookieGreaterEqual(logic.WebRequest)
logic.TestGreaterEqualTag.testCookieGreaterEqual()
logic.TestGreaterThanTag.beginCookieGreaterThan(logic.WebRequest)
logic.TestGreaterThanTag.testCookieGreaterThan()
logic.TestNotEqualTag.beginCookieStringEquals(logic.WebRequest)
logic.TestNotEqualTag.testCookieStringEquals()
logic.TestNotPresentTag.beginCookiePresent(logic.WebRequest)
logic.TestNotPresentTag.testCookiePresent()
logic.TestPresentTag.beginCookiePresent(logic.WebRequest)
logic.TestPresentTag.testCookiePresent()

case of test suite deletions), I excluded them from analysis, and cannot speak to whether these represent binary incompatibilities in STRUTS-TILES framework, the example code, or the test suite, nor whether HIMA's analysis is correct. The Corpus only considered changes to the STRUTS API itself.

D.5.3 Non-API Changes Considered by Meng et al.

Table D.19: Meng et al. Matched API Changes. (All in org.apache.struts)

Original	Replacement
action.ActionServlet.parseModuleConfigFile(java.lang.String, java.lang.String, action.ModuleConfig, action.Digester, java.lang.String)	action.ActionServlet.parseModuleConfigFile(action.Digester, java.lang.String)
config.AddDataSourcePropertyRule.begin(org.xml.sax.Attributes)	config.AddDataSourcePropertyRule.begin(java.lang.String, java.lang.String, org.xml.sax.Attributes)
config.PlugInSetPropertyRule.begin(org.xml.sax.Attributes)	config.PlugInSetPropertyRule.begin(java.lang.String, java.lang.String, org.xml.sax.Attributes)
config.SetActionMappingClassRule.begin(org.xml.sax.Attributes)	config.SetActionMappingClassRule.begin(java.lang.String, java.lang.String, org.xml.sax.Attributes)
taglib.html.JavascriptValidatorTag.getStartElement()	taglib.html.JavascriptValidatorTag.renderStartElement()
tiles.EmptyIterator.hasNext()	java.util.Collections.EMPTY_LIST.iterator().hasNext()
tiles.EmptyIterator.next()	java.util.Collections.EMPTY_LIST.iterator().next()
tiles.EmptyIterator.remove()	java.util.Collections.EMPTY_LIST.iterator().remove()

Table D.19 lists changes that HIMA detects in the STRUTS library implementation of entities that do not form part of its API. As with the AURA tool in Section D.5.3, HIMA also identifies changes to entities that have **private** or **package** visibility, and thus are by definition not part of the library's API. It is worth noting that the adaptations HIMA detects are correct, and that it also correctly resolves three cases that AURA does not, involving the removal of the EmptyIterator nested class. As with Section D.3.2 and Section D.5.3, the Corpus does not record these as changes to the STRUTS API.

Table D.20 lists methods with **private** visibility, and thus are not part of the API, that are reported as deleted by HIMA. The Corpus does not record such changes, as the API is meant to provide library developers with the flexibility to modify and alter such functionality without af-

Table D.20: Meng et al. Deleted methods with **private** visibility in the STRUTS library (All in org.apache.struts).

Original
taglib.html.JavascriptValidatorTag.getNextVar(java.lang.String)
taglib.html.JavascriptValidatorTag.replaceChar(java.lang.String, **int**, **char**)
util.RequestUtils.retrieveMessageResources(util.PageContext, java.lang.String, **boolean**)

fecting code dependent on its interface; as such, I cannot say whether replacement functionality does or does not exist for these methods.

D.5.4 Changes on which the Corpus and Meng et al. Disagree

There are three categories of changes which the HIMA tool identifies in the STRUTS 1.1.0 to 1.2.4 API migration that disagree with the findings in the Corpus.

1. **Identifying a change and replacement in the API, when no change occurred.** Table D.21 presents a case where HIMA identified a change occurring in the STRUTS API between version 1.1.0 and 1.2.4 that the Corpus does not. On investigation, there is no actual change to the API; however, it appears that some difference in the method triggered one of HIMA's analyses, causing it to report a change rule where no change has occurred.

2. **Identifying API functionality as deleted, when it was actually refactored in a binary compatible fashion.** Table D.21 presents cases where HIMA identified **public** methods as being deleted without replacement functionality; because these methods are part of the STRUTS API, their deletion should be noted in the Corpus, but the Corpus makes no mention of either method being altered in a binary incompatible fashion. On investigation, I determined that this is because these methods were not deleted: rather, both methods were moved into the superclass of their containing class, which is otherwise called a "pull-up method" refactoring [Fowler,

1999].

3. **Identifying functionality as deleted, when replacement functionality existed within the STRUTS library.** Table D.23 lists methods in the STRUTS API that were identified as being deleted without replacement by HIMA, and for which the Corpus indicates a replacement exists within STRUTS.

Table D.21: Meng et al. Unchanged method reported as changed (All in org.apache.struts).

Original
tiles.ComponentDefinition.ComponentDefinition(tiles.xmlDefinition.XmlDefinition)

Table D.22: Meng et al. Methods marked deleted, that were instead refactored into their superclass in the STRUTS API (All in org.apache.struts).

Original
validator.DynaValidatorActionForm.validate(validator.ActionMapping, validator.HttpServletRequest)
validator.ValidatorActionForm.validate(validator.ActionMapping, validator.HttpServletRequest)

D.5.5 Comparison with Meng et al.: Summary

Table D.24 summarizes the comparison of the HIMA tool's identified change rules, to those of the Corpus for the STRUTS v1.1.0 to v1.2.4 migration. Of the 107 changes HIMA identifies, 32 concern binary incompatible changes of which all are in agreement with the Corpus. The remaining changes detected concern internal functionality in the STRUTS library's implementation, or to test suites and example programs that accompany the library.

However, many of the method deletions detected by HIMA proved erroneous. Only 3 such deletions were correctly detected: in these cases, replacement functionality was provided through external libraries that HIMA could not analyze, and their approach appropriately allowed for such cases. While many of the detected deletions concerned modifications to methods with **private** or **package** visibility, or to its test suites and example programs, nearly half of the detected method

deletions involved API methods for which the Corpus indicates a replacement within the STRUTS API exists.

D.6 Summary: Comparing Existing Work to the Library Migration Corpus

Dig et al. [2006], Schäfer et al. [2008], Wu et al. [2010], and Meng et al. [2012] have previously used the STRUTS v1.1.0 to 1.2.4 migration to evaluate the accuracy of their approaches for detecting changes in software libraries and APIs. However, comparing their results to the binary incompatible changes recorded in the Library Migration Corpus showed significant differences in both the changes reported as having occurred in the STRUTS API, and their identified replacement.

Table D.25 summarizes my investigation of the changes identified by REFACTORINGCRAWLER, the approach of Schäfer et al. [2008], AURA, and HIMA for changes in the STRUTS library. Most of the changes reported by these techniques did not involve binary incompatible changes to the API: some reported changes to the STRUTS test-suite, or a co-located library that offered related functionality; others reported changes caused by backwards compatible refactorings, or deprecated code that would some day be removed.

In cases where REFACTORINGCRAWLER, the approach of Schäfer et al. [2008], AURA, and HIMA reported replacements for binary incompatible changes to the STRUTS API, their results confirmed those of the Corpus, and in a few cases provided alternative replacements. In cases where there was disagreement, a detailed investigation showed the Corpus to be correct in every case.

```
1  /**
2   * Encapsulate a collection of ActionFormBean objects that can be
3   * administered and searched, while hiding the internal implementation.
4   *
5   * @author Craig R. McClanahan
6   * @version Revision : 1.8 Date : 2003/04/1500 : 18 : 45
7   *
8   * @deprecated Replaced by collection of FormBeanConfig instances
9   * in {@link org.apache.struts.config.ModuleConfig}
10  */
11
12 public class ActionFormBeans implements Serializable {
13      /**
14       * The collection of ActionFormBean instances, keyed by name.
15       */
16      protected FastHashMap formBeans = new FastHashMap();
17 //Code elided
18      /**
19       * Register a form bean to the set configured for this servlet.
20       *
21       * @param formBean The formBean to be added
22       */
23      public void addFormBean(ActionFormBean formBean) {
24      formBeans.put(formBean.getName(), formBean);
25      }
26      /**
27       * Return the formBean associated with the specified logical name,
28       * if any; otherwise return <code>null</code>.
29       *
30       * @param name Logical name of the desired form bean
31       */
32      public ActionFormBean findFormBean(String name) {
33      return ((ActionFormBean) formBeans.get(name));
34      }
35      /**
36       * Return the set of names for form beans defined in this collection.
37       * If there are no such formBeans, a zero-length array is returned.
38       */
39      public String[] findFormBeans() {
40          return
41            ((String[]) formBeans.keySet().toArray(new String[formBeans.size()]));
42      }
43      /**
44       * Deregister a formBean from the set configured for this servlet.
45       *
46       * @param formBean The formBean to be deregistered
47       */
48      public void removeFormBean(ActionFormBean formBean) {
49      formBeans.remove(formBean.getName());
50      }
```

Listing D.24: The ActionFormBeans class, v1.1.0

```
1  public class ModuleConfigImpl implements Serializable, ModuleConfig {
2  //Code elided
3      /**
4       * Add a new <code>FormBeanConfig</code> instance to the set associated
5       * with this module.
6       *
7       * @param config The new configuration instance to be added
8       *
9       * @exception java.lang.IllegalStateException if this module configuration
10      * has been frozen
11      */
12     public void addFormBeanConfig(FormBeanConfig config) {
13         if (configured) {
14             throw new IllegalStateException("Configuration is frozen");
15         }
16         formBeans.put(config.getName(), config);
17     }
18 //Code elided
19     /**
20      * Return the form bean configuration for the specified key, if any;
21      * otherwise return <code>null</code>.
22      *
23      * @param name Name of the form bean configuration to return
24      */
25     public FormBeanConfig findFormBeanConfig(String name) {
26         return ((FormBeanConfig) formBeans.get(name));
27     }
28     /**
29      * Return the form bean configurations for this module. If there
30      * are none, a zero-length array is returned.
31      */
32     public FormBeanConfig[] findFormBeanConfigs() {
33         FormBeanConfig results[] = new FormBeanConfig[formBeans.size()];
34         return ((FormBeanConfig[]) formBeans.values().toArray(results));
35     }
36 //Code elided
37     /**
38      * The set of form bean configurations for this module, if any,
39      * keyed by the <code>name</code> property.
40      */
41     protected HashMap formBeans = null;
42  //Code elided
43     /**
44      * The default class name to be used when creating action form bean
45      * instances.
46      */
47     protected String actionFormBeanClass = "org.apache.struts.action.ActionFormBean";
48 //Code elided
49 }
```

Listing D.25: The ModuleConfigImpl class, v1.2.4

```
1
2
3 public void checkBeanLocale( ConfigHelper website_config ) {
4
5   ActionFormBeans beans_map    = website_config.getActionFormBeans();
6   ActionFormBean  loginform_userid = beans_map.findFormBean( "login.userid" );
7   ActionFormBean  loginform_passwd  = beans_map.findFormBean( "login.passwd" );
8   //Code elided
9 }
```

Listing D.26: Example using the ConfigHelper.getActionFormBeans() method, v1.1.0

```
1  public void checkBeanLocale( ModuleConfig website_module_config ) {
2
3    FormBeanConfig     beans_set[]   = website_module_config.findFormBeanConfigs();
4    HashMap<String,ActionFormBean> beans_map   = new HashMap<String,ActionFormBean>();
5    for ( FormBeanConfig current_bean : beans_set ) {
6      ActionFormBean casted_bean = ( ActionFormBean )current_bean;
7      beans_map.put( casted_bean.getName(), casted_bean );
8    }
9
10   ActionFormBean      loginform_userid = beans_map.get( "login.userid" );
11   ActionFormBean      loginform_passwd  = beans_map.get( "login.passwd" );
12   //Code elided
13 }
```

Listing D.27: Example using the replacement ModuleConfig.findFormBeanConfigs() method, v1.2.4

Table D.23: Meng et al. Methods marked deleted, for which replacements existed in the STRUTS API (All in org.apache.struts).

Original
config.ConfigHelper.errorOutput()
config.ConfigHelper.errorOutput(java.lang.String)
config.ConfigHelper.errors()
config.ConfigHelper.errors(java.lang.String)
config.ConfigHelper.errorsEmpty()
config.ConfigHelper.errorSize()
config.ConfigHelper.errorSize(java.lang.String)
config.ConfigHelper.getActionErrors()
config.ConfigHelper.getActionFormBeans()
config.ConfigHelper.getActionForwards()
config.ConfigHelper.getActionMappings()
config.ConfigHelper.getErrorOutput()
config.ConfigHelper.getErrorOutput(java.lang.String)
config.ConfigHelper.getErrors()
config.ConfigHelper.getErrors(java.lang.String)
config.ConfigHelper.getErrorsEmpty()
config.ConfigHelper.getErrorSize()
config.ConfigHelper.getErrorSize(java.lang.String)
config.ConfigHelperInterface.getActionErrors()
config.ConfigHelperInterface.getActionFormBeans()
config.ConfigHelperInterface.getActionForwards()
config.ConfigHelperInterface.getActionMappings()
config.ConfigHelperInterface.getErrorOutput()
config.ConfigHelperInterface.getErrorOutput(java.lang.String)
config.ConfigHelperInterface.getErrors()
config.ConfigHelperInterface.getErrors(java.lang.String)
config.ConfigHelperInterface.getErrorsEmpty()
config.ConfigHelperInterface.getErrorSize()
config.ConfigHelperInterface.getErrorSize(java.lang.String)
config.ConfigHelperInterface.getLocale()
config.FormBeanConfig.getModuleConfig()
config.FormBeanConfig.setModuleConfig(config.ModuleConfig)
taglib.html.FormTag.getName()
taglib.html.FormTag.getScope()
taglib.html.FormTag.getType()
taglib.html.FormTag.setName(java.lang.String)
taglib.html.FormTag.setScope(java.lang.String)
taglib.html.FormTag.setType(java.lang.String)

Table D.24: Comparison of HIMA to the Corpus for STRUTS 1.1.0 to 1.2.4.

Changes Identified	BI Changes		Non-BI Changes		Non-STRUTS
	Agrees	Disagrees	Correct	Error	
107	32	0	8	1	66
Deletions Identified	Replacement Exists		Non-BI	Non-STRUTS	Not Deleted
	In API	External			
77	38	3	3	31	2

Table D.25: Comparison of all techniques against the Corpus for STRUTS 1.1.0 to 1.2.4.

Approach	Changes Identified	BI Changes		Non-BI Changes		Non-STRUTS Changes
		Correct	Error	Correct	Error	
REFACTORINGCRAWLER	65	20	0	25	3	17
Schäfer et al. [2008]	66	10	0	56	0	0
AURA	135	34	38	10	3	50
HIMA	187	35	40	11	1	97

Appendix E

The Library Migration Corpus

In this appendix, I present version 2.0 of the Library Migration Corpus (the "Corpus"). The Corpus represents the complete set of binary incompatible changes made to the APIs of the examined Java libraries, over the versions considered. The details of the methodology used to identify binary incompatibilities, and investigate their replacements, are presented in Section 5.1.

The following JAVA libraries, and their APIs, were examined.

APACHE LUCENE versions 3.1.0, v3.2.0, v3.3.0, v3.4.0;

APACHE STRUTS[1] versions 1.0.2, 1.1.0, 1.2.4, 1.2.6, 1.2.7, 1.2.8, 1.2.9.;

DBCP[2] versions 1.0.0, 1.1.0, 1.2.0, 1.2.1, 1.2.2, 1.3.0, 1.4.0;

JDOM[3] versions 1.0.b6, 1.0.b7, 1.0.b8, 1.0.b9;

LOG4J[4] versions 1.0.4, 1.1.3, 1.2.1, 1.2.2, 1.2.5, 1.2.6, 1.2.7, 1.2.8, 1.2.9; and,

SLF4J[5] versions 1.0.1, 1.5.1, 1.5.5, 1.5.8.

E.1 Notation and Formatting

To simplify the recording of class, method, and field identifiers, and to add lexical cues that would help easily distinguish them from each other, the following notation has been used.

Classes. Class names are recorded using their fully qualified JAVA name, e.g., java.lang.String. Nested classes are noted by placing a dollar sign ("$") between the enclosing class's

[1] http://struts.apache.org/
[2] http://commons.apache.org/proper/commons-dbcp/
[3] http://www.jdom.org/
[4] http://logging.apache.org/log4j/1.2/
[5] http://www.slf4j.org/

fully qualified name, and the name of the nested class, e.g., org.apache.lucene.index.MergePolicy$OneMerge.

Methods. Methods are recorded by using the fully qualified JAVA name of the enclosing class, a period (".") to denote the start of the method name, and the method's identifier. Additionally, the parameter list is recorded within parentheses ("()"), as well as the types of any parameters required—parameter names are not recorded. The return type is recorded within brackets ("[]") at the end of the parameter list.[6] The following two methods are different examples of this notation.

- org.apache.struts.action.Action.getResources()[MessageResources]
- org.apache.struts.action.Action.perform(ActionMapping, ActionForm,-ServletRequest, ServletResponse)[ActionForward]

Class constructors are treated as a version of a method, with the class's name used as the method name, and no return type being provided. For example, org.apache.struts.action.ActionMessages$ActionMessageItem.ActionMessageItem(List, **int**) describes the constructor for the nested ActionMessageItem class.

Because I was manually recording these entities, I omit the fully qualified name from types in the parameter list and return types, and simply use the type's identifier: otherwise, the lengths of the resulting identifiers could be so long that visually comparing the BI entity and its replacement would be impossible. While this raises the potential of confusion, it was not an issue during this study.

Fields. Fields are recorded using the fully qualified JAVA name of the enclosing class, a colon (":") to denote the start of the field's identifier, and the field's identifier. The field's type is recorded using square brackets ("[]") after the field name.

[6] Strictly speaking, it is not necessary to record return types: the rules of overloading declarations in JAVA do not allow two methods with identical names and parameter lists, but different return types, to exist. Because of this, recording the return type is redundant in identifying the method.

Generics and types in arrays are recording using the normal JAVA syntax, with the exception that arrays are always associated with the type name.

E.2 Table Column Explanations

The column labels in each table, and their meaning, are as follows.

Original. The identifier of the API entity in the old version of the library that is binary incompatible with the new library version. If "none" appears, this typically means a new method was introduced into a JAVA **interface**.

Replacement(s). The identifier of the API entity or entities in the new version of the library that I determined offer replacement functionality, or equivalent functionality.

Because not all replacements are 1:1 in nature, the records in this column have to accommodate the nature of the transformation. If multiple entities are needed (i.e., it is a 1:n change), these are separated with a semicolon (";"). If multiple original entities are replaced with one entity (i.e., it is an n:1 change), the same replacement will show up for each of the associated "original entities". If multiple are replaced with multiple (i.e., it is a m:n change), the two are combined: the semicolon separated list of replacement entities appears at all locations of the associated "old entities" they replace.[7]

BI Code and BI Description. The CLIRR tool provides a summary of each binary incompatible change between each version, giving an error code that classifies the error, and a

[7] Admittedly, this can be a point of confusion in recording and representing the nature of the transformation involved in an API change. This particular notation evolved out of initially using a spreadsheet to record the list of binary incompatible changes identified in the API; all identified transformations were then indexed by the BI entity involved. I suggest that a better format for storing this data would be something like an XML tag/tree data structure, which could allow for a more precise specification of 1:n, n:1, and m:n changes. This would make distinguishing between cases where several entities have the same replacement (i.e., 1:1 replacements but with the same entity), and where several entities are replaced by one (i.e., n:1 replacements) easier in the dataset.

short textual description. While the BI description will be more meaningful to the reader, I have left in the BI code as well.

Vis. This records whether the visibility of the entity is **public** or **protected**. The CLIRR tool does not provide this information, so it was recorded separately using the QDOX tool.

Depr? This is short for "Was this method deprecated prior to becoming a binary incompatibility?" If "Y", a "@deprecate" JAVADOC tag was associated with the old entity in the previous version of the source code.

Xform. The nature of the transformation the replacement signifies: 1:1, 1:n, n:1, m:n, or 1:0 to mark deletions.

Classif. This records the classification of the API change, as described in Section 5.1.3. If multiple classifications were deemed appropriate, they were separated with a semicolon.

Notes. These are notes I wrote during my investigation on how this entity had changed, and what its replacement entails. If there are no notes, I deemed it to be a trivial change.

E.3 The Library Migration Corpus

E.3.1 APACHE LUCENE

The 3.1.0 to 3.2.0 migration is presented in Table E.1, 3.2.0 to 3.3.0 in Table E.2, and 3.3.0 to 3.4.0 in Table E.3.

E.3.2 APACHE STRUTS

The 1.0.2 to 1.1.0 migration is presented in Table E.4, 1.1.0 to 1.2.4 in Table E.5, 1.2.4 to 1.2.6 in Table E.6, and 1.2.6 to 1.2.7 in Table E.7. Note that for the APACHE STRUTS 1.2.7 to 1.2.8, and the 1.2.8 to 1.2.9 migrations, there are no binary incompatible changes to the API; thus, a table does not appear for those migrations.

E.3.3 DBCP

None of the migrations examined contained any binary incompatible changes; thus tables do not appear for this library.

E.3.4 JDOM

The b6 to b7 migration is presented in Table E.8, b7 to b8 in Table E.9, and b8 to b9 in Table E.10.

E.3.5 LOG4J

The 1.0.4 to 1.1.3 migration is presented in Table E.11, 1.1.3 to 1.2.1 in Table E.12, 1.2.1 to 1.2.2 in Table E.13, 1.2.4 to 1.2.5 in Table E.14, 1.2.5 to 1.2.6 in Table E.15, and 1.2.8 to 1.2.9 in Table E.16. Note that for the LOG4J 1.2.2 to 1.2.4, and the 1.2.6 through 1.2.8 migrations, there are no binary incompatible changes to the API; thus, a table does not appear for those migrations.

E.3.6 SLF4J

None of the migrations examined contained any binary incompatible changes; thus tables do not appear for this library.

Table E.1: APACHE LUCENE 3.1.0 to 3.2.0 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index.MergePolicy$OneMerge: segments[SegmentInfos]	org.apache.lucene.index.MergePolicy$OneMerge: segments[List<SegmentInfo>]	6004	Changed type of field segments from org.apache.lucene.index.SegmentInfos to java.util.List	**public**	N	1:1	Remove Wrapper	
org.apache.lucene.index.MergePolicy$OneMerge.OneMerge(SegmentInfos)	org.apache.lucene.index.MergePolicy$OneMerge.OneMerge(List<SegmentInfo>)	7005	Parameter 1 of '**public** MergePolicy$OneMerge(org.apache.lucene.index.SegmentInfos)' has changed its type to java.util.List	**public**	N	1:n	Change Type	
org.apache.lucene.index.SegmentInfos	org.apache.lucene.index.SegmentInfos.asList()[List<SegmentInfo>]; org.apache.lucene.index.SegmentInfos.asSet()[Set<SegmentInfo>]	4001	Removed java.io.Serializable from the set of implemented interfaces	**public**	N	1:n	Supertype Change; Partial Encapsulation	The original class (which is **final**) extended Vector. Now it doesn't, but contains a List datatype within. You're now expected to use an asList() call to get a list of the data which can be used in place of Vector. However, the lists are now unmodifiable, so this is partially encapsulated.

Continued on next page

Table E.1 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index.SegmentInfos	org.apache.lucene.index.SegmentInfos.asList()[List<SegmentInfo>]; org.apache.lucene.index.SegmentInfos.asSet()[Set<SegmentInfo>]	4001	Removed java.util.Collection from the set of implemented interfaces	**public**	N	1:n	Supertype Change; Partial Encapsulation	The original class (which is **final**) extended Vector. Now it doesn't, but contains a List datatype within. You're now expected to use an asList() call to get a list of the data which can be used in place of Vector. However, the lists are now unmodifiable, so this is partially encapsulated.
org.apache.lucene.index.SegmentInfos	org.apache.lucene.index.SegmentInfos.asList()[List<SegmentInfo>]; org.apache.lucene.index.SegmentInfos.asSet()[Set<SegmentInfo>]	4001	Removed java.util.List from the set of implemented interfaces	**public**	N	1:n	Supertype Change; Partial Encapsulation	The original class (which is **final**) extended Vector. Now it doesn't, but contains a List datatype within. You're now expected to use an asList() call to get a list of the data which can be used in place of Vector. However, the lists are now unmodifiable, so this is partially encapsulated.

Continued on next page

Table E.1 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index.SegmentInfos	org.apache.lucene.index.SegmentInfos.asList()[List<SegmentInfo>]; org.apache.lucene.index.SegmentInfos.asSet()[Set<SegmentInfo>]	4001	Removed java.util.RandomAccess from the set of implemented interfaces	**public**	N	1:n	Supertype Change; Partial Encapsulation	The original class (which is **final**) extended Vector. Now it doesn't, but contains a List datatype within. You're now expected to use an asList() call to get a list of the data which can be used in place of Vector. However, the lists are now unmodifiable, so this is partially encapsulated.
org.apache.lucene.index.SegmentInfos	org.apache.lucene.index.SegmentInfos.asList()[List<SegmentInfo>]; org.apache.lucene.index.SegmentInfos.asSet()[Set<SegmentInfo>]	5001	Removed java.util.AbstractCollection from the list of superclasses	**public**	N	1:n	Supertype Change; Partial Encapsulation	The original class (which is **final**) extended Vector. Now it doesn't, but contains a List datatype within. You're now expected to use an asList() call to get a list of the data which can be used in place of Vector. However, the lists are now unmodifiable, so this is partially encapsulated.

Continued on next page

Table E.1 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index.SegmentInfos	org.apache.lucene.index.SegmentInfos.asList()[List<SegmentInfo>]; org.apache.lucene.index.SegmentInfos.asSet()[Set<SegmentInfo>]	5001	Removed java.util.AbstractList from the list of superclasses	**public**	N	1:n	Supertype Change; Partial Encapsulation	The original class (which is **final**) extended Vector. Now it doesn't, but contains a List datatype within. You're now expected to use an asList() call to get a list of the data which can be used in place of Vector. However, the lists are now unmodifiable, so this is partially encapsulated.
org.apache.lucene.index.SegmentInfos	org.apache.lucene.index.SegmentInfos.asList()[List<SegmentInfo>]; org.apache.lucene.index.SegmentInfos.asSet()[Set<SegmentInfo>]	5001	Removed java.util.Vector from the list of superclasses	**public**	N	1:n	Supertype Change; Partial Encapsulation	The original class (which is **final**) extended Vector. Now it doesn't, but contains a List datatype within. You're now expected to use an asList() call to get a list of the data which can be used in place of Vector. However, the lists are now unmodifiable, so this is partially encapsulated.

Continued on next page

Table E.1 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index. SegmentReader.getNorms(String)[**byte**[]]	org.apache.lucene.index. SegmentReader:norms[Map< String,SegmentNorms>]; java.util.Map<String, SegmentNorms>.get(String)[SegmentNorms]	7002	Method '**protected byte**[] getNorms(java.lang.String)' has been removed	protected	N	1:n	Expose/Unpack Implementa- tion; Change Type	The type change here is trivial as the classes are largely the same, but a manual specification of their sameness will still be needed, or at least a forced type cast, or alterations of developer code that used the old type.
org.apache.lucene.index. SegmentWrite. StateSegmentWriteState(PrintStream, Directory, String, FieldInfos, **int**, **int**)	org.apache.lucene.index. BufferedDeletes; org.apache.lucene.index. SegmentWrite. StateSegmentWriteState(PrintStream, Directory, String, FieldInfos, **int**, **int**, BufferedDeletes)	7004	In method '**public** SegmentWriteState(java. io.PrintStream, org.apache. lucene.store.Directory, java.lang. String, org.apache.lucene.index. FieldInfos, **int**, **int**)' the number of arguments has changed	**public**	N	1:1	Add Parameter	This is the Add Parameter refactoring, as defined in Fowler's book. The code that relies on this parameter checks for null, so a null value is appropriate to supply here (which Fowler suggests).

Continued on next page

Table E.1 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.util.PriorityQueue:heap[T*[]]	org.apache.lucene.util.PriorityQueue.getHeapArray()[Object[]]	6010	Accessibility of field heap has been weakened from **protected** to **private**	**protected**	N	1:n	Hide; Encapsulate; Generics Conversion	The original field is now **private**, but a getter method is provided—since the getter is providing a reference to the original field array, it is modifiable, and thus acts like a setter as well. There is a Generics Conversion here as well: the original method is a generic type, but this returns an object. A type cast will be needed where the type is pre-known!

Table E.2: APACHE LUCENE 3.2.0 to 3.3.0 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index. LogMergePolicy.isOptimized(SegmentInfos, **int**, Set$< $SegmentInfo>)[**boolean**]	org.apache.lucene.index. LogMergePolicy.isOptimized(SegmentInfos, **int**, Map< SegmentInfo,Boolean>)[**boolean**]	7005	Parameter 3 of 'protected **boolean** isOptimized(org.apache.lucene.index. SegmentInfos, **int**, java.util.Set)' has changed its type to java.util.Map	**protected**	N	1:n	Change Type	Essentially pairs a Boolean object attribute with each entry in the original set; this Boolean is used by other parts of the program to track whether modifications have been made that mean this Segment shouldn't be optimized. At minimum, the replacement is adding this additional initialization step of creating a Boolean.TRUE value with each SegmentInfo entry.

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index.LogMergePolicy.findMergesForOptimize(SegmentInfos, **int**, Set<SegmentInfo>)[MergePolicy$MergeSpecification]	org.apache.lucene.index.LogMergePolicy.findMergesForOptimize(SegmentInfos, **int**, Map<SegmentInfo,Boolean>)[MergePolicy$MergeSpecification]	7005	Parameter 3 of 'public org.apache.lucene.index.MergePolicy$MergeSpecification findMergesForOptimize(org.apache.lucene.index.SegmentInfos, **int**, java.util.Set)' has changed its type to java.util.Map	public	N	1:n	Change Type	This is a change to a parent class that trickles down to its children in the API. Essentially pairs a Boolean object attribute with each entry in the original set; this Boolean is used by other parts of the program to track whether modifications have been made that mean this Segment shouldn't be optimized. At minimum, the replacement is adding this additional initialization step of creating a Boolean.TRUE value with each SegmentInfo entry.

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index. MergePolicy.findMergesFor Optimize(SegmentInfos, **int**, Set <SegmentInfo>)[MergePolicy $MergeSpecification]	org.apache.lucene.index. MergePolicy.findMergesFor Optimize(SegmentInfos, **int**, Map <SegmentInfo,Boolean>)[MergePolicy$MergeSpecification]	7005	Parameter 3 of 'public org.apache.lucene.index. MergePolicy$MergeSpecification findMergesForOptimize(org. apache.lucene.index. SegmentInfos, **int**, java.util.Set)' has changed its type to java.util.Map	public	N	1:n	Change Type	This is a change to a parent class that trickles down to its children in the API. Essentially pairs a Boolean object attribute with each entry in the original set; this Boolean is used by other parts of the program to track whether modifications have been made that mean this Segment shouldn't be optimized. At minimum, the replacement is adding this additional initialization step of creating a Boolean.TRUE value with each SegmentInfo entry.

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index. NoMergePolicy.findMergesFor Optimize(SegmentInfos, **int**, Set <SegmentInfo>)[MergePolicy $MergeSpecification]	org.apache.lucene.index. NoMergePolicy.findMergesFor Optimize(SegmentInfos, **int**, Map <SegmentInfo,Boolean>)[MergePolicy$MergeSpecification]	7005	Parameter 3 of 'public org.apache.lucene.index. MergePolicy$MergeSpecification findMergesForOptimize(org. apache.lucene.index. SegmentInfos, **int**, java.util.Set)' has changed its type to java.util.Map	public	N	1:n	Change Type	This is a change to a parent class that trickles down to its children in the API. Essentially pairs a Boolean object attribute with each entry in the original set; this Boolean is used by other parts of the program to track whether modifications have been made that mean this Segment shouldn't be optimized. At minimum, the replacement is adding this additional initialization step of creating a Boolean.TRUE value with each SegmentInfo entry.

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index. TieredMergePolicy.findMergesFor Optimize(SegmentInfos, **int**, Set <SegmentInfo>)[MergePolicy $MergeSpecification]	org.apache.lucene.index. TieredMergePolicy.findMergesFor Optimize(SegmentInfos, **int**, Map <SegmentInfo,Boolean>)[MergePolicy$MergeSpecification]	7005	Parameter 3 of 'public org.apache.lucene.index. MergePolicy$MergeSpecification findMergesForOptimize(org. apache.lucene.index. SegmentInfos, **int**, java.util.Set)' has changed its type to java.util.Map	public	N	1:n	Change Type	This is a change to a parent class that trickles down to its children in the API. Essentially pairs a Boolean object attribute with each entry in the original set; this Boolean is used by other parts of the program to track whether modifications have been made that mean this Segment shouldn't be optimized. At minimum, the replacement is adding this additional initialization step of creating a Boolean.TRUE value with each SegmentInfo entry.

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.index. UpgradeIndexMergePolicy. findMergesForOptimize(SegmentInfos, **int**, Set< SegmentInfo>)[MergePolicy $MergeSpecification]	org.apache.lucene.index. UpgradeIndexMergePolicy. findMergesForOptimize(SegmentInfos, **int**, Map< SegmentInfo,Boolean>)[MergePolicy$MergeSpecification]	7005	Parameter 3 of 'public org.apache.lucene.index. MergePolicy$MergeSpecification findMergesForOptimize(org. apache.lucene.index. SegmentInfos, **int**, java.util.Set)' has changed its type to java.util.Map	public	N	1:n	Change Type	This is a change to a parent class that trickles down to its children in the API. Essentially pairs a Boolean object attribute with each entry in the original set; this Boolean is used by other parts of the program to track whether modifications have been made that mean this Segment shouldn't be optimized. At minimum, the replacement is adding this additional initialization step of creating a Boolean.TRUE value with each SegmentInfo entry.
org.apache.lucene.search. FieldComparator.value(**int**)[Comparable<T>]	org.apache.lucene.search. FieldComparator.value(**int**)[T]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.Object	public	N	1:1	Generics Change	This is a change of the Generics restriction in this method. Because it was Comparable<?>, it restricted the kinds of methods it could return to those implementing Comparable. Now, it only returns the Generic Type instantiated with the object.

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.search.FieldComparator$ByteComparator.value(**int**)[Comparable<?>]	org.apache.lucene.search.FieldComparator$ByteComparator.value(**int**)[Byte]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.Byte	public	N	1:1	Remove Generics	
org.apache.lucene.search.FieldComparator$DocComparator.value(**int**)[Comparable<?>]	org.apache.lucene.search.FieldComparator$DocComparator.value(**int**)[Integer]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.Integer	public	N	1:1	Remove Generics	
org.apache.lucene.search.FieldComparator$DoubleComparator.value(**int**)[Comparable<?>]	org.apache.lucene.search.FieldComparator$DoubleComparator.value(**int**)[Double]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.Double	public	N	1:1	Remove Generics	
org.apache.lucene.search.FieldComparator$FloatComparator.value(**int**)[Comparable<?>]	org.apache.lucene.search.FieldComparator$FloatComparator.value(**int**)[Float]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.Float	public	N	1:1	Remove Generics	
org.apache.lucene.search.FieldComparator$IntComparator.value(**int**)[Comparable<?>]	org.apache.lucene.search.FieldComparator$IntComparator.value(**int**)[Integer]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.Integer	public	N	1:1	Remove Generics	

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.search. FieldComparator $LongComparator.value(**int**)[Comparable<?>]	org.apache.lucene.search. FieldComparator $LongComparator.value(**int**)[Long]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.Long	**public**	N	1:1	Remove Generics	
org.apache.lucene.search. FieldComparator $RelevanceComparator.value(**int**) [Comparable<?>]	org.apache.lucene.search. FieldComparator $RelevanceComparator.value(**int**) [Float]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.Float	**public**	N	1:1	Remove Generics	
org.apache.lucene.search. FieldComparator $ShortComparator.value(**int**)[Comparable<?>]	org.apache.lucene.search. FieldComparator $ShortComparator.value(**int**)[Short]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.Short	**public**	N	1:1	Remove Generics	
org.apache.lucene.search. FieldComparator $StringComparatorLocale.value(**int**)[Comparable<?>]	org.apache.lucene.search. FieldComparator $StringComparatorLocale.value(**int**)[String]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.String	**public**	N	1:1	Remove Generics	
org.apache.lucene.search. FieldComparator$StringOrdVal Comparator.value(**int**)[Comparable<?>]	org.apache.lucene.search. FieldComparator$StringOrdVal Comparator.value(**int**)[String]	7006	Return type of method 'public java.lang.Comparable value(**int**)' has been changed to java.lang.String	**public**	N	1:1	Remove Generics	

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.search.FieldComparator$StringValComparator.value(**int**)[Comparable<?>]	org.apache.lucene.search.FieldComparator$StringValComparator.value(**int**)[String]	7006	Return type of method '**public** java.lang.Comparable value(**int**)' has been changed to java.lang.String	**public**	N	1:1	Remove Generics	
org.apache.lucene.search.FieldDoc:fields[Comparable[]]	org.apache.lucene.search.FieldDoc:fields[Object[]]	6004	Changed type of field fields from java.lang.Comparable[] to java.lang.Object[]	**public**	N	1:1	Change Type	Removes a required interface, and just accepts a generic object now.
org.apache.lucene.search.FieldDoc.FieldDoc((**int**, **float**, Comparable[])	org.apache.lucene.search.FieldDoc.FieldDoc((**int**, **float**, Object[])	7005	Parameter 3 of '**public** FieldDoc(**int**, **float**, java.lang.Comparable[])' has changed its type to java.lang.Object[]	**public**	N	1:1	Change Type	Removes a required interface, and just accepts a generic object now.
org.apache.lucene.search.MultiTermQuery.rewrite(IndexReader)[Query]	org.apache.lucene.search.MultiTermQuery.rewrite(IndexReader)[Query] or org.apache.lucene.search.MultiTermQuerygetEnum(IndexReader)[FilteredTermEnum]	7014	Method '**public** org.apache.lucene.search.Query rewrite(org.apache.lucene.index.IndexReader)' is now **final**	**public**	N	1:1; 1:n	Immutable	This is tricky. For callers, the method is unchanged. For subclasses that override the method, it's much more problematic (thus the 1:n). The documentation reports that this can be worked around by getting an enum of the query terms, and then implementing your own rewriter. But it's not entirely clear how to do that.

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.search.Query.weight(Searcher)[Weight]	org.apache.lucene.search.Query.weight(Searcher)[Weight] or org.apache.lucene.search.Query.createWeight(Searcher)[Weight]	7014	Method '**public** org.apache.lucene.search.Weight weight(org.apache.lucene.search.Searcher)' is now **final**	**public**	N	1:1; 1:n	Immutable; Pre-Deprecation Crippling	This is tricky. For callers, the method is unchanged. For subclasses that override the method, it's much more problematic (thus the 1:n). The documentation reports that this can be worked around by using another method, but that method just throws an exception, meaning it needs to be overridden and reimplemented.
org.apache.lucene.search.Searcher.createWeight(Query)[Weight]	org.apache.lucene.search.Searcher.createWeight(Query)[Weight] or org.apache.lucene.search.Searcher.createNormalizedWeight(Query)[Weight]	7014	Method '**protected** org.apache.lucene.search.Weight createWeight(org.apache.lucene.search.Query)' is now **final**	**protected**	N	1:1	Immutable; Pre-Deprecation Crippling	Immutable method is just delegated anyways, easy to override the delegated method instead
org.apache.lucene.store.IndexOutput.copyBytes(IndexInput, **long**)[**void**]	org.apache.lucene.store.DataOutput.copyBytes(DataInput, **long**)[**void**]	7002	Method '**public void** copyBytes(org.apache.lucene.store.IndexInput, **long**)' has been removed	**public**	N	1:1	Change Type	Replaced by its supertype, but it doesn't seem to have been using any of the methods in the subtype anyways. Also note the move of the method up the inheritance hierarchy.

Continued on next page

Table E.2 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.store.MMapDirectory.getMaxChunkSize()[**int**]	org.apache.lucene.store.MMapDirectory.getMaxChunkSize()[**int**]	7014	Method '**public int** getMaxChunkSize()' is now **final**	**public**	N	1:1	Immutable	Subclasses could just inline this functionality to override/modify it, as it's quite small.
org.apache.lucene.store.MMapDirectory.setMaxChunkSize(**int**)[**void**]	org.apache.lucene.store.MMapDirectory.setMaxChunkSize(**int**)[**void**]	7014	Method '**public void** setMaxChunkSize(**int**)' is now **final**	**public**	N	1:1; 1:n	Immutable	Overriding this is difficult as the internal implementation is designed to prevent you from altering their modification that chunk sizes must be powers of 2.
org.apache.lucene.store.RAMOutputStream.copyBytes(IndexInput, **long**)[**void**]	org.apache.lucene.store.RAMOutputStream.copyBytes(DataInput, **long**)[**void**]	7005	Parameter 1 of '**public void** copyBytes(org.apache.lucene.store.IndexInput, **long**)' has changed its type to org.apache.lucene.store.DataInput	**public**	N	1:1	Change Type	Replaced by its supertype, but it doesn't seem to have been using any of the methods in the subtype anyways. Unlike its parent class change of this method, the replacement is found in the same class in the new API.
org.apache.lucene.util.IOUtils.closeSafely(Closeable[])[**void**]	org.apache.lucene.util.IOUtils.closeSafely(Exception, Closeable ...)[Exception]	7004	In method '**public void** closeSafely(java.io.Closeable[])' the number of arguments has changed	**public**	N	1:1	Add Parameter	This is a Fowler "Add Parameter" refactoring, where **null** is an acceptable value and causes it to behave as before.

Table E.3: APACHE LUCENE 3.3.0 to 3.4.0 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.document.AbstractField: omitTermFreqAndPositions[**boolean**]	org.apache.lucene.document.AbstractField:indexOptions[IndexOptions]	6001	Removed field omitTermFreqAndPositions	**protected**	N	1:n	Change Type	Type is changed to an enum; library usage examples show how to convert the type to a **boolean**.
org.apache.lucene.document.Fieldable.getOmitTermFreqAndPositions()[**boolean**]	org.apache.lucene.document.Fieldable.getIndexOptions()[IndexOptions]	7002	Method '**public boolean** getOmitTermFreqAndPositions()' has been removed	**public**	N	1:n	Change Type; Rename	
org.apache.lucene.document.Fieldable.getIndexOptions()[IndexOptions]	none	7012	Method '**public** org.apache.lucene.index.FieldInfo$IndexOptions getIndexOptions()' has been added to an interface	**public**	N	1:n	Introduce Replacement	
org.apache.lucene.document.Fieldable.setIndexOptions(IndexOptions)[**void**]	none	7012	Method '**public void** setIndexOptions(org.apache.lucene.index.FieldInfo$IndexOptions)' has been added to an interface	**public**	N	1:n	Introduce Replacement	
org.apache.lucene.document.Fieldable.setOmitTermFreqAndPositions(**boolean**)[**void**]	org.apache.lucene.document.Fieldable.setIndexOptions(IndexOptions)[**void**]	7002	Method '**public void** setOmitTermFreqAndPositions(**boolean**)' has been removed	**public**	N	1:n	Change Type; Rename	

Continued on next page

Table E.3 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.search.FieldCache.getUnValuedDocs(IndexReader, String)[DocIdSet]	none	7012	Method '**public** org.apache.lucene.search.DocIdSet getUnValuedDocs(org.apache.lucene.index.IndexReader, java.lang.String)' has been added to an interface	**public**	N	1:n	Add Method to Interface	
org.apache.lucene.util.BitVector.subset(**int**, **int**)[BitVector]	org.apache.lucene.util.BitVector.get(**int**)[**boolean**]; org.apache.lucene.util.BitVector	7002	Method '**public** org.apache.lucene.util.BitVector subset(**int**, **int**)' has been removed	**public**	N	1:n		Will need to manually call the get() method to get each individual bit, then use the BitVector type to encapsulate it.
org.apache.lucene.util.IOUtils.closeSafely(**boolean**, Closeable[])[**void**]	org.apache.lucene.util.IOUtils.closeWhileHandlingException(**boolean**, Closeable)[**void**] or org.apache.lucene.util.IOUtils.close(Closeablele)[**void**]	7002	Method '**public void** closeSafely(**boolean**, java.io.Closeable[])' has been removed	**public**	N	1:1	Remove parameter; OR Rename operation.	Depending on the replacement chosen, which are both viable, the nature of the transformation is different.
org.apache.lucene.util.IOUtils.closeSafely(**boolean**, Iterable)[**void**]	org.apache.lucene.util.IOUtils.closeWhileHandlingException(**boolean**, Iterable)[**void**]	7002	Method '**public void** closeSafely(**boolean**, java.lang.Iterable)' has been removed	**public**	N	1:1	Rename	

Continued on next page

Table E.3 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.lucene.util.IOUtils.closeSafely(Exception, Closeable[])[**void**]	org.apache.lucene.util.IOUtils.closeWhileHandlingException(Exception, Closeable[])[**void**]	7002	Method '**public void** closeSafely(java.lang.Exception, java.io.Closeable[])' has been removed	**public**	N	1:1	Rename	
org.apache.lucene.util.IOUtils.closeSafely(Exception, Iterable)[**void**]	org.apache.lucene.util.IOUtils.closeSafely(Exception, Iterable)[**void**]	7002	Method '**public void** closeSafely(java.lang.Exception, java.lang.Iterable)' has been removed	**public**	N	1:1	Rename	
org.apache.lucene.util.SortedVIntList.SortedVIntList(OpenBitSet)	org.apache.lucene.util.SortedVIntList.SortedVIntList(BitSet)	7002	Method '**public** SortedVIntList(org.apache.lucene.util.OpenBitSet)' has been removed	**public**	N	1:n	Change Type; Replace with External API	Changed from using an internal type, to a type from the Java SDK.
org.apache.lucene.util.fst.Builder.Builder(INPUT_TYPE, **int**, **int**, **boolean**, Outputs)	org.apache.lucene.util.fst.Builder.Builder(INPUT_TYPE, **int**, **int**, **boolean**, **boolean**, **int**, Outputs<T>)	7004	In method '**public** Builder(org.apache.lucene.util.fst.FST$INPUT_TYPE, **int**, **int**, **boolean**, org.apache.lucene.util.fst.Outputs)' the number of arguments has changed	**public**	N	1:1	Add parameter	Again, the add parameter refactoring. Examining the other constructor, which delegates to this one, suggests default values which can be supplied.

Table E.4: APACHE STRUTS 1.0.2 to 1.1.0 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.Action.perform(ActionServlet, ActionMapping, ActionForm, ServletRequest, ServletResponse)[ActionForward]	org.apache.struts.action.Action.perform(ActionMapping, ActionForm, ServletRequest, ServletResponse)[ActionForward]]	7002	Method '**public** org.apache.struts.action.ActionForward perform(org.apache.struts.action.ActionServlet, org.apache.struts.action.ActionMapping, org.apache.struts.action.ActionForm, javax.servlet.ServletRequest, javax.servlet.ServletResponse)' has been removed	**public**	Y	1:1	Remove parameter	Parameter deleted. Delegated in previous versions. Was deprecated before removed.
org.apache.struts.action.Action.perform(ActionServlet, ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)[ActionForward]	org.apache.struts.action.Action.perform(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)[ActionForward]	7002	Method '**public** org.apache.struts.action.ActionForward perform(org.apache.struts.action.ActionServlet, org.apache.struts.action.ActionMapping, org.apache.struts.action.ActionForm, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)' has been removed	**public**	Y	1:1	Remove parameter	Parameter deleted. Delegated in previous versions. Was deprecated before removed.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionBase	org.apache.struts.action.Action	8001	Class org.apache.struts.action.ActionBase removed	**public**	Y	1:1	Shorten subtype chain	Subclass of Action, just shortens the hierarchy. I gather that this was intended to be a useful, abstract subclass with some default utility methods, but it has been moved into the superclass. Was deprecated
org.apache.struts.action.ActionErrors:errors[HashMap]	org.apache.struts.action.ActionMessages:messages[HashMap]	6001	Removed field errors	**protected**	N	1:1	Pull up	Moved into superclass; HashMap now incorporates ActionMessages, of which ActionErrors (which this field was to represent) are a subclass. Code comment in the original is largely the same as in the new version, but the type/class name referred to in the comment was changed.
org.apache.struts.action.ActionForm.validate()[String[]]	org.apache.struts.action.ActionForm.validate(ActionMapping, HttpServletRequest)[ActionErrors]	7002	Method '**public** java.lang.String [] validate()' has been removed	**public**	Y	1:n	Add contextual data	Was deprecated

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionFormBeans:formBeans[FastHashMap]	org.apache.struts.action.ActionFormBeans:formBeans[FastHashMap]	6004	Changed type of field formBeans from org.apache.struts.util.FastHashMap to org.apache.commons.collections.FastHashMap	**protected**	N	1:1	Extract to external API	Interesting. The type has changed, because they moved it into an Apache Commons Library. But it looks like it's the same I think. So this would require an import update. Release documentation notes this change, but doesn't note where it occurs.
org.apache.struts.action.ActionForward:name[String]	org.apache.struts.config.ForwardConfig:name[String]	6001	Removed field name	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields. Note that the usage of the methods which rely on this field is unchanged. It would be interesting if SEMDIFF-styled tools can pick this up.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionForward:path[String]	org.apache.struts.config.ForwardConfig:path[String]	6001	Removed field path	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields. Note that the usage of the methods which rely on this field is unchanged as it is a source compatible, but binary incompatible change. It would be interesting if SEMDIFF-styled tools can pick this up.
org.apache.struts.action.ActionForward:redirect[**boolean**]	org.apache.struts.config.ForwardConfig:redirect[**boolean**]	6001	Removed field redirect	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields. Note that the usage of the methods which rely on this field is unchanged as it is a source compatible, but binary incompatible change. It would be interesting if SEMDIFF-styled tools can pick this up.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping:attribute[String]	org.apache.struts.config.ActionConfig:attribute[String]	6001	Removed field attribute	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields. Note that the usage of the methods which rely on this field is unchanged as it is a source compatible, but binary incompatible change. It would be interesting if SEMDIFF-styled tools can pick this up.
org.apache.struts.action.ActionMapping:forward[String]	org.apache.struts.config.ActionConfig:forward[String]	6001	Removed field forward	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields. Note that the usage of the methods which rely on this field is unchanged as it is a source compatible, but binary incompatible change. It would be interesting if SEMDIFF-styled tools can pick this up.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping:forwards[ActionForwards]	org.apache.struts.config.ActionConfig:forwards[HashMap]	6001	Removed field forwards	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields. Note that the usage of the methods which rely on this field is unchanged as it is a source compatible, but binary incompatible change. It would be interesting if SEMDIFF-styled tools can pick this up.
org.apache.struts.action.ActionMapping:include[String]	org.apache.struts.config.ActionConfig:include[String]	6001	Removed field include	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields. Note that the usage of the methods which rely on this field is unchanged as it is a source compatible, but binary incompatible change. It would be interesting if SEMDIFF-styled tools can pick this up.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping:input[String]	org.apache.struts.config.ActionConfig:input[String]	6001	Removed field input	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields. Note that the usage of the methods which rely on this field is unchanged as it is a source compatible, but binary incompatible change. It would be interesting if SEMDIFF-styled tools can pick this up.
org.apache.struts.action.ActionMapping:instance[Action]	none	6001	Removed field instance	**protected**	N	Delete	Delete	Was not deprecated, no replacement offered. No calls to it within the current class, not sure what this was phased out in favour of.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping:mappings[ActionMappings]	org.apache.struts.config.impl.ModuleConfigImpl:actionConfigs[HashMap]	6001	Removed field mappings	**protected**	N	1:n	Expose/ Unpack Implementation; Remap	Javadoc comment for the ActionMappings class in the library says that it is being replaced by a collection of ActionConfigs. This would be the closest equivalent. Additional calls will be needed to make the HashMap $\Longrightarrow$ ActionConfig equivalent to ActionMappings $\Longrightarrow$ ActionMapping. Note that a search on the method signature will yield three matches in reasonable replacements in the new library.
org.apache.struts.action.ActionMapping:multipartClass[String]	org.apache.struts.config.ActionConfig.multipartClass[String]	6001	Removed field multipartClass	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.
org.apache.struts.action.ActionMapping:name[String]	org.apache.struts.config.ActionConfig.name[String]	6001	Removed field name	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping:parameter[String]	org.apache.struts.config.ActionConfig.parameter[String]	6001	Removed field parameter	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.
org.apache.struts.action.ActionMapping:path[String]	org.apache.struts.config.ActionConfig.path[String]	6001	Removed field path	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.
org.apache.struts.action.ActionMapping:prefix[String]	org.apache.struts.config.ActionConfig.prefix[String]	6001	Removed field prefix	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.
org.apache.struts.action.ActionMapping:scope[String]	org.apache.struts.config.ActionConfig.scope[String]	6001	Removed field scope	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping:suffix[String]	org.apache.struts.config.ActionConfig.suffix[String]	6001	Removed field suffix	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.
org.apache.struts.action.ActionMapping:type[String]	org.apache.struts.config.ActionConfig.type[String]	6001	Removed field type	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.
org.apache.struts.action.ActionMapping:unknown[**boolean**]	org.apache.struts.config.ActionConfig.unknown[**boolean**]	6001	Removed field unknown	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.
org.apache.struts.action.ActionMapping:validate[**boolean**]	org.apache.struts.config.ActionConfig.validate[**boolean**]	6001	Removed field validate	**protected**	N	1:1	Pull Up	Created a superclass and moved the fields up. Noticed that the new class version had added a supertype, examining the supertype led me to find the fields.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping.getActionClass()[String]	org.apache.struts.action.ActionConfig.getType()[String]	7002	Method 'public java.lang.String getActionClass()' has been removed	public	Y	1:1	Rename; Move	Deprecated before deleted.
org.apache.struts.action.ActionMapping.getFormAttribute()[String]	org.apache.struts.action.ActionConfig.getAttribute()[String]	7002	Method 'public java.lang.String getFormAttribute()' has been removed	public	Y	1:1	Rename; Move	
org.apache.struts.action.ActionMapping.getFormClass()[String]	org.apache.struts.config.FormBeanConfig.getName()[String]	7002	Method 'public java.lang.String getFormClass()' has been removed	public	Y	1:1	Rename; Move	Was deprecated, and delegated in old version. Notes indicate to replace with a method call on a ActionFormBean class, but the current delegation matches behaviour in the new library, which is this method, so the comment notes seem to be wrong.
org.apache.struts.action.ActionMapping.getFormPrefix()[String]	org.apache.struts.action.ActionConfig.getPrefix()[String]	7002	Method 'public java.lang.String getFormPrefix()' has been removed	public	Y	1:1	Rename; Move	
org.apache.struts.action.ActionMapping.getFormScope()[String]	org.apache.struts.action.ActionConfig.getScope()[String]	7002	Method 'public java.lang.String getFormScope()' has been removed	public	Y	1:1	Rename; Move	

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping.getFormSuffix()[String]	org.apache.struts.action.ActionConfig.getSuffix()[String]	7002	Method '**public** java.lang.String getFormSuffix()' has been removed	**public**	Y	1:1	Rename; Move	
org.apache.struts.action.ActionMapping.getInputForm()[String]	org.apache.struts.action.ActionConfig.getInput()[String]	7002	Method '**public** java.lang.String getInputForm()' has been removed	**public**	Y	1:1	Rename; Move	
org.apache.struts.action.ActionMapping.createFormInstance()[ActionForm]	org.apache.struts.config.impl.ModuleConfigImpl.findFormBeanConfig(String)[FormBeanConfig]; org.apache.struts.config.FormBeanConfig.getName()[String]; java.lang.Class.forName(String)[Class]; java.lang.Class.newInstance()[ActionForm]	7002	Method '**public** org.apache.struts.action.ActionForm createFormInstance()' has been removed	**public**	Y	1:n	Expose/Unpack Implementation	Comment states that the controller servlet needs to do this now, but provides no details on how to do that. This is taken from the existing implementation, which still works in the new library.
org.apache.struts.action.ActionMapping.getMappings()[ActionMappings]	org.apache.struts.config.impl.ModuleConfigImpl.findActionConfigs()[ActionConfig[]]	7002	Method '**public** org.apache.struts.action.ActionMappings getMappings()' has been removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Was not deprecated. Getter deleted, along with the associated field.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping.addForward(ActionForward)[**void**]	org.apache.struts.config.ActionConfig:addForwardConfig(ForwardConfig)[**void**]	7002	Method '**public void** addForward(\code{org.apache.struts.action.ActionForward)' has been removed	**public**	N	1:1	Move; Rename; Change Type	Was not deprecated, no replacement offered. ForwardConfig is a supertype of ActionForward. The code comment that lead me to realize this replacement was found in the class comment in ActionForward in the new version.
org.apache.struts.action.ActionMapping.removeForward(ActionForward)[**void**]	org.apache.struts.config.ActionConfig:removeForwardConfig(ForwardConfig)[**void**]	7002	Method '**public void** removeForward(org.apache.struts.action.ActionForward)' has been removed	**public**	N	1:1	Move; Rename; Change Type	Was not deprecated, no replacement offered. ForwardConfig is a supertype of ActionForward. The code comment that lead me to realize this replacement was found in the class comment in ActionForward in the new version.
org.apache.struts.action.ActionMapping.setActionClass(String)[**void**]	org.apache.struts.action.ActionConfig.setType(String)[**void**]	7002	Method '**public void** setActionClass(java.lang.String)' has been removed	**public**	Y	1:1	Rename; Move	Was deprecated, and delegated in old version
org.apache.struts.action.ActionMapping.setFormAttribute(String)[**void**]	org.apache.struts.action.ActionConfig.setAttribute(String)[**void**]	7002	Method '**public void** setFormAttribute(java.lang.String)' has been removed	**public**	Y	1:1	Rename; Move	Was deprecated, and delegated in old version

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping.setFormClass(String)[**void**]	org.apache.struts.config.FormBeanConfig.setName(String)[**void**]	7002	Method '**public void** setFormClass(java.lang.String)' has been removed	**public**	Y	1:1	Rename; Move	Was deprecated, and delegated in old version. Notes indicate to replace with a method call on a ActionFormBean class, but the current delegation matches behaviour in the new library, which is this method, so the comment notes seem to be wrong.
org.apache.struts.action.ActionMapping.setFormPrefix(String)[**void**]	org.apache.struts.action.ActionConfig.setPrefix(String)[**void**]	7002	Method '**public void** setFormPrefix(java.lang.String)' has been removed	**public**	Y	1:1	Rename; Move	Was deprecated, and delegated in old version
org.apache.struts.action.ActionMapping.setFormScope(String)[**void**]	org.apache.struts.action.ActionConfig.setScope(String)[**void**]	7002	Method '**public void** setFormScope(java.lang.String)' has been removed	**public**	Y	1:1	Rename; Move	Was deprecated, and delegated in old version
org.apache.struts.action.ActionMapping.setFormSuffix(String)[**void**]	org.apache.struts.action.ActionConfig.setSuffix(String)[**void**]	7002	Method '**public void** setFormSuffix(java.lang.String)' has been removed	**public**	Y	1:1	Rename; Move	Was deprecated, and delegated in old version
org.apache.struts.action.ActionMapping.setInputForm(String)[**void**]	org.apache.struts.action.ActionConfig.setInput(String)[**void**]	7002	Method '**public void** setInputForm(java.lang.String)' has been removed	**public**	Y	1:1	Rename; Move	Was deprecated, and delegated in old version

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMapping.setMappings(ActionMappings)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.addActionConfig(ActionConfig)[]	7002	Method '**public void** setMappings(org.apache.struts.action.ActionMappings)' has been removed	**public**	N	1:n	Move; Rename; Change Type	Deleted, wasn't deprecated. Calls exist to this (and sister method removeMapping) in the ActionMappings class, but are commented out in the new version. The code comment that directed me to the ModuleConfig/ModuleConfigImpl class was found in the class comment in ActionMappings in the new version. so not easily found.
org.apache.struts.action.ActionMappingBase	org.apache.struts.action.ActionMapping	8001	Class org.apache.struts.action.ActionMappingBase removed	**public**	Y	1:1	Shorten subtype chain	Deleted, was deprecated. Apparently its superclass was formerly an interface, and is now a class, so there's no need for this. The abstract supertype becomes a class, and the subclass is no longer needed.
org.apache.struts.action.ActionMappings:mappings[FastHashMap]	org.apache.struts.action.ActionMappings:mappings[FastHashMap]	6004	Changed type of field mappings from org.apache.struts.util.FastHashMap to org.apache.commons.collections.FastHashMap	**protected**	N	1:1	Extract to external API	Type is basically the same, but class was moved into an external library, so its qualified name changed.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet:actions[FastHashMap]	org.apache.struts.action.RequestProcessor:actions[FastHashMap]	6001	Removed field actions	**protected**	N	1:n	Extract field; Change Type	Deleted, wasn't deprecated. The comment associated with this field in the old and new version is the same, and is the only occurrence of that comment in the code.
org.apache.struts.action.ActionServlet:application[MessageResources]	org.apache.struts.action.Action.getResources(HttpServletRequest)[MessageResources]; javax.servlet.HttpServletRequest.setAttribute(Globals.MESSAGES_KEY, MessageResources)[**void**]	6001	Removed field application	**protected**	N	1:n	Expose/Unpack Implementation	Deleted, wasn't deprecated. Three methods using these field were also deleted. This field was moved into a ServletRequest attribute, and needs to be accessed through getters/setters now.
org.apache.struts.action.ActionServlet:bufferSize[**int**]	org.apache.struts.config.ControllerConfig:bufferSize[**int**]	6001	Removed field bufferSize	**protected**	N	1:1	Move	Deleted, wasn't deprecated. Three methods using these field were also deleted. I found this by searching for the string literal associated with the ActionServlet.content field, and also found this field moved into the same class that was moved into.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet:content[String]	org.apache.struts.config.ControllerConfig:contentType[String]	6001	Removed field content	**protected**	N	1:1	Move; Rename	Deleted, wasn't deprecated. Three methods using these field were also deleted. Found my searching for matches on the string literal that initializes this field.
org.apache.struts.action.ActionServlet:dataSources[FastHashMap]	org.apache.struts.action.ActionServlet:dataSources[FastHashMap]	6004	Changed type of field dataSources from org.apache.struts.util.FastHashMap to org.apache.commons.collections.FastHashMap	**protected**	N	1:1	Extract to external API	Type is basically the same, but class was moved into an external library.
org.apache.struts.action.ActionServlet:defaultLocale[Locale]	java.util.Locale.getDefault()[Locale]	6001	Removed field defaultLocale	**protected**	N	1:1	Expose/Unpack Implementation	Deleted, wasn't deprecated. Field wasn't referenced within the class, and was basically delegating to the result of Locale.getDefault().

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet:factoryClass[String]	org.apache.struts.config.MessageResourcesConfig:factory[String]	6001	Removed field factoryClass	**protected**	N	1:n	Move; Rename; Encapsulate	Deleted, wasn't deprecated. Found this by looking for equivalent replacements for the callers of this field in the new version, and seeing one replaced with this. This also makes sense with the overall changes to the library, as they've moved from system-wide settings to more module-specific settings, which this is. Getters and setters will now need to be used to interact with this.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet:formBeanClass[String]	org.apache.struts.config.impl.ModuleConfigImpl:formBeans[HashMap]	6001	Removed field formBeanClass	**protected**	N	n:m	Consolidate	Deleted, wasn't deprecated. The ModuleConfigImpl offers a replacement for formBeanClass and formBeans that instead uses a HashMap field. Since ActionFormBeans itself was a wrapper for a collection of FormBeans, the collection stored in the HashMap is the same, although some additional calls need to be made to properly convert them. 2 fields replaced by one field, but some additional calls need to be made before they're equivalent.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet:formBeans[ActionFormBeans]	org.apache.struts.config.impl.ModuleConfigImpl:formBeans[HashMap]	6001	Removed field formBeans	**protected**	N	n:m	Consolidate	Deleted, wasn't deprecated. The ModuleConfigImpl offers a replacement for formBeanClass and formBeans that instead uses a HashMap field. Since ActionFormBeans itself was a wrapper for a collection of FormBeans, the collection stored in the HashMap is the same, although some additional calls need to be made to properly convert them. 2 fields replaced by one field, but some additional calls need to be made before they're equivalent.
org.apache.struts.action.ActionServlet:forwardClass[String]	org.apache.struts.config.impl.ModuleConfigImpl:forwards[HashMap]	6001	Removed field forwardClass	**protected**	N	n:m	Consolidate	Deleted, wasn't deprecated. Five methods using these field were also deleted.
org.apache.struts.action.ActionServlet:forwards[ActionForwards]	org.apache.struts.config.impl.ModuleConfigImpl:forwards[HashMap]	6001	Removed field forwards	**protected**	N	n:m	Consolidate	Deleted, wasn't deprecated. Five methods using these field were also deleted.
org.apache.struts.action.ActionServlet:locale[**boolean**]	org.apache.struts.config.ControllerConfig:locale[**boolean**]	6001	Removed field locale	**protected**	N	1:1	Move	Deleted, wasn't deprecated. Two methods using these field were also deleted.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet:mappingClass[String]	org.apache.struts.config.impl.ModuleConfigImpl:actionMappingClass[String]	6001	Removed field mappingClass	**protected**	N	1:1	Move; Rename	Deleted, wasn't deprecated.
org.apache.struts.action.ActionServlet:mappings[ActionMappings]	org.apache.struts.config.impl.ModuleConfigImpl:actionConfigs[HashMap]	6001	Removed field mappings	**protected**	N	n:m	Expose/Unpack Implementation; Remap	Deleted, wasn't deprecated. The ModuleConfigImpl offers a replacement for that uses a HashMap field; it's meant to store ActionConfig objects, but since ActionMapping is a subtype of ActionConfig, those are legal as well. Again, ActionMappings are a collection of ActionMapping, so some additional work will be needed to convert between the two.
org.apache.struts.action.ActionServlet:maxFileSize[String]	org.apache.struts.config.ControllerConfig:maxFileSize[String]	6001	Removed field maxFileSize	**protected**	N	1:1	Move	

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet:multipartClass[String]	org.apache.struts.config.ControllerConfig:multipartClass[String]	6001	Removed field multipartClass	**protected**	N	1:1	Move	Deleted, wasn't deprecated. Three methods in the class using these field were also deleted, as were a method in another class in the same package. Lexical search yields the ActionConfig and ControllerConfig classes as likely replacements. Since ActionConfig is meant to be module-wide, and ControllerConfig is meant to be element-wide, and since most other deletions here were also moved into the ControllerConfig class, I'm going to reason that ControllerConfig is meant to be the more appropriate replacement here.
org.apache.struts.action.ActionServlet:nocache[**boolean**]	org.apache.struts.config.ControllerConfig:nocache[**boolean**]	6001	Removed field nocache	**protected**	N	1:1	Move	Deleted, wasn't deprecated. Two methods using these field were also deleted.
org.apache.struts.action.ActionServlet:tempDir[String]	org.apache.struts.config.ControllerConfig:tempDir[String]	6001	Removed field tempDir	**protected**	N	1:1	Move	Deleted, wasn't deprecated. Two methods using these field were also deleted.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet:validate[**boolean**]	org.apache.struts.config.ActionConfig:validate[**boolean**]	6001	Removed field validate	**protected**	N	1:1	Move	Deleted, wasn't deprecated. Two methods using these field were also deleted.
org.apache.struts.action.ActionServlet:validating[**boolean**]	javax.servlet.GenericServlet.getServletConfig()[ServletConfig]; javax.servlet.ServletConfig.getInitParameter(String)[String]; java.lang.Boolean.parseBoolean(String)[**boolean**]	6001	Removed field validating	**protected**	N	1:n	Extract to external API; Encapsulate	Deleted, wasn't deprecated. Two methods using these field were also deleted. Found by doing a lexical search for the type/field name in the new version, noticed that it was changed into a local field within a method; in that same method body, the field was used to call the appropriate getter/setter on the Digester class.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processForward(ActionMapping, HttpServletRequest, HttpServletResponse)[**boolean**]	org.apache.struts.action.RequestProcessor.processForward(HttpServletRequest, HttpServletResponse, ActionMapping)[**boolean**]	7002	Method '**protected boolean** processForward(org.apache.struts.action.ActionMapping, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)' has been removed	**protected**	N	1:1	Move; Parameters reordered	Moved method into a newly created class, promoted visibility to **public**, changed order of parameters. Found by tracking the method which called this, matching it in the new version, noticing that numerous calls had been replaced by a single call chain, going through that call chain until I came to the RequestProcessor class, which then contained this and several other moved methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processInclude(ActionMapping, HttpServletRequest, HttpServletResponse)[**boolean**]	org.apache.struts.action.RequestProcessor.processInclude(HttpServletRequest, HttpServletResponse, ActionMapping)[**boolean**]	7002	Method '**protected boolean** processInclude(org.apache.struts.action.ActionMapping, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)' has been removed	**protected**	N	1:1	Move; Parameters reordered	Moved method into a newly created class, promoted visibility to **public**, changed order of parameters. Found by tracking the method which called this, matching it in the new version, noticing that numerous calls had been replaced by a single call chain, going through that call chain until I came to the RequestProcessor class, which then contained this and several other moved methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processPreprocess(HttpServletRequest, HttpServletResponse)[**boolean**]	org.apache.struts.action.RequestProcessor.processPreprocess(HttpServletRequest, HttpServletResponse)[**boolean**]	7002	Method '**protected boolean** processPreprocess(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)' has been removed	**protected**	N	1:1	Move; Parameters reordered	Moved method into a newly created class, promoted visibility to **public**, changed order of parameters. Found by tracking the method which called this, matching it in the new version, noticing that numerous calls had been replaced by a single call chain, going through that call chain until I came to the RequestProcessor class, which then contained this and several other moved methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processValidate(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)[**boolean**]	org.apache.struts.action.RequestProcessor.processValidate(HttpServletRequest, HttpServletResponse, ActionForm, ActionMapping)[**boolean**]	7002	Method '**protected boolean** processValidate(org.apache.struts.action.ActionMapping, org.apache.struts.action.ActionForm, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)' has been removed	**protected**	N	1:1	Move; Parameters reordered	Moved method into a newly created class, promoted visibility to **public**, changed order of parameters. Found by tracking the method which called this, matching it in the new version, noticing that numerous calls had been replaced by a single call chain, going through that call chain until I came to the RequestProcessor class, which then contained this and several other moved methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processPath(HttpServletRequest)[String]	org.apache.struts.action.RequestProcessor.processPath(HttpServletRequest)[String]	7002	Method '**protected** java.lang.String processPath(javax.servlet.http.HttpServletRequest)' has been removed	**protected**	N	1:1	Move; Parameters reordered	Moved method into a newly created class, promoted visibility to **public**, changed order of parameters. Found by tracking the method which called this, matching it in the new version, noticing that numerous calls had been replaced by a single call chain, going through that call chain until I came to the RequestProcessor class, which then contained this and several other moved methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action. ActionServlet. processActionCreate(ActionMapping, HttpServletRequest)[Action]	org.apache.struts.action. RequestProcessor. processActionCreate(HttpServletRequest, HttpServletResponse, ActionMapping)[Action]	7002	Method '**protected** org.apache. struts.action.Action processActionCreate(org.apache. struts.action.ActionMapping, javax.servlet.http. HttpServletRequest)' has been removed	**protected**	N	1:1	Move; Parameters reordered	Moved method into a newly created class, promoted visibility to **public**, changed order of parameters. Found by tracking the method which called this, matching it in the new version, noticing that numerous calls had been replaced by a single call chain, going through that call chain until I came to the RequestProcessor class, which then contained this and several other moved methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processActionForm(ActionMapping, HttpServletRequest)[ActionForm]	org.apache.struts.action.RequestProcessor.processActionForm(HttpServletRequest, HttpServletResponse, ActionMapping)[ActionForm]	7002	Method '**protected** org.apache.struts.action.ActionForm processActionForm(org.apache.struts.action.ActionMapping, javax.servlet.http.HttpServletRequest)' has been removed	**protected**	N	1:1	Move; Parameters reordered	Moved method into a newly created class, promoted visibility to **public**, changed order of parameters. Found by tracking the method which called this, matching it in the new version, noticing that numerous calls had been replaced by a single call chain, going through that call chain until I came to the RequestProcessor class, which then contained this and several other moved methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processActionPerform(Action, ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)[ActionForward]	org.apache.struts.action.RequestProcessor.processActionPerform(HttpServletRequest, HttpServletResponse, Action, ActionForm, ActionMapping)[ActionForward]	7002	Method '**protected** org.apache.struts.action.ActionForward processActionPerform(org.apache.struts.action.Action, org.apache.struts.action.ActionMapping, \code{org.apache.struts.action.ActionForm, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)' has been removed	**protected**	N	1:1	Move; Parameters reordered	Moved method into a newly created class, promoted visibility to **public**, changed order of parameters. Found by tracking the method which called this, matching it in the new version, noticing that numerous calls had been replaced by a single call chain, going through that call chain until I came to the RequestProcessor class, which then contained this and several other moved methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processMapping(String, HttpServletRequest)[ActionMapping]	org.apache.struts.action.RequestProcessor.processMapping(HttpServletRequest, HttpServletResponse, String)[ActionMapping]	7002	Method '**protected** org.apache.struts.action.ActionMapping processMapping(java.lang.String, javax.servlet.http.HttpServletRequest)' has been removed	**protected**	N	1:1	Move; Parameters reordered	Moved method into a newly created class, promoted visibility to **public**, changed order of parameters. Found by tracking the method which called this, matching it in the new version, noticing that numerous calls had been replaced by a single call chain, going through that call chain until I came to the RequestProcessor class, which then contained this and several other moved methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.initDigester(**int**)[Digester]	org.apache.struts.action.ActionServlet.initConfigDigester()[Digester]; org.apache.commons.digester.Digester.setDebug(**int**)[**void**]	7002	Method '**protected** org.apache.struts.digester.Digester initDigester(**int**)' has been removed	**protected**	N	1:1	Rename; Extract to external API; Expose/Unpack Implementation	Deleted, wan't deprecated. Note that the return type Digester was moved into an external library; the parameter supplied in the old version was the debugging level, which now needs to be specified on the Digester object. Found by tracing callers of this method up to the init() method in ActionServlet, comparing to the init() method in the new library, and noticing a call to destroyDigester() which seemed to deal with the same type. Finding that lead me to find the new Digester field, and find this method. Comments in old and new methods suggest that this is the replacement. Release notes also point out that Digester is now part of the apache.commons library, confirming this change.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.initDigesterOld(**int**)[Digester]	org.apache.struts.action.ActionServlet.initConfigDigester()[Digester]	7002	Method '**protected** org.apache.struts.digester.Digester initDigesterOld(**int**)' has been removed	**protected**	N	1:n	Remap; Remove Parameter	Deleted, wan't deprecated. Comments and implementation indicate that this method was meant to support an old file configuration format; with this method deleted the format isn't accessible at all by the API. Seems to be to force migration to the new format. Note that the Remap part of the transformation is that configuration files also need to change.
org.apache.struts.action.ActionServlet.destroyActions()[**void**]	org.apache.struts.action.RequestProcessor.destroy()[**void**]	7002	Method '**protected void** destroyActions()' has been removed	**protected**	N	1:1	Move; Rename	Moved into a newly created class, and changed the name. Functionality is the same. Found by previously knowing how the associated field for this method was replaced, and seeing what methods used that new, replacement field. Implementations are highly similar.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.destroyApplication() [**void**]	javax.servlet.ServletRequest.removeAttribute(Globals.MESSAGES_KEY)[**void**]	7002	Method '**protected void** destroyApplication()' has been removed	**protected**	N	1:1	Expose/Unpack Implementation	Deleted, wasn't deprecated. The field this method was setting to null has been moved into a ServletRequest attribute, and so is external to this class, and isn't needed to be released by the class. Tracing the Action.MESSAGES_KEY in the new version lead to the realization that Global.MESSAGES_KEY was the replacement, and that it should be removed in this fashion. The reason I'm not 100% sure is that I don't know if this was deleted to prevent the attribute from being removed, and instead leaving it up to the servlet container to remove the attribute on shutdown, rather than having the program do it?

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.initActions()[**void**]	org.apache.struts.action.RequestProcessor.init(ActionServlet, ModuleConfig)[**void**]	7002	Method '**protected void** initActions()' has been removed	**protected**	N	n:1	Move; Add contextual data; Encapsulate	Deleted, wasn't deprecated. The field this method was initializing was extracted into a new class, and this is the equivalent method. Note that this method initializes a few more things than the original did.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.initApplication()[**void**]	org.apache.struts.action.ActionServlet.initModuleMessageResources(ModuleConfig)[**void**]	7002	Method '**protected void** initApplication()' has been removed	**protected**	N	1:1	Remap; Add contextual data	Deleted, wasn't deprecated. Note that if you check what this method's callers replace it with, you'll be directed to initServlet() which isn't a match. To get this, I needed to trace up the use of a the main type in the implementation of this method, MessageResourcesFactory, and see that it was used only in one method in the original class, and only in one method in the new class. Further confirming this is that another method called initApplicationMessageResources(ModuleConfig) was added, immediately deprecated, and delegated to this method. The developers clearly meant that pseudo-method to be a signpost from the old method to this method.
org.apache.struts.action.ActionServlet.initDebug()[**void**]	org.apache.struts.action.ActionServlet.initOther()[**void**]	7002	Method '**protected void** initDebug()' has been removed	**protected**	N	n:1	Consolidate	Deleted, wasn't deprecated; functionality merged into another method.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.initMapping()[**void**]	org.apache.struts.action.ActionServlet.init()[**void**]	7002	Method '**protected void** initMapping()' has been removed	**protected**	N	n:1	Consolidate	Deleted, wasn't deprecated. The method was radically rewritten, and incorporated into another method which also does other initializations. Instead of initializing a preset set of attributes as before, it now iterates through all configuration attributes associated with the servlet, and passes them to another method which sorts our how to use that attribute to initialize a ModuleConfig for those values.
org.apache.struts.action.ActionServlet.initUpload()[**void**]	org.apache.struts.action.ActionServlet.defaultControllerConfig(ModuleConfig)[**void**]	7002	Method '**protected void** initUpload()' has been removed	**protected**	N	1:1	Change Type	Delete, wasn't deprecated. Found by lexically searching string literals and their enclosing call chain. The replacement is deprecated though; while it's clear they want to configure a ControllerConfig object instead of this approach, it's not clear how to do that yet.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processActionForward(ActionForward, ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)[**void**]	org.apache.struts.action.RequestProcessor.processActionForward(HttpServletRequest, HttpServletResponse, ActionForward)[**void**]	7002	Method '**protected void** processActionForward(org.apache.struts.action.ActionForward, org.apache.struts.action.ActionMapping, org.apache.struts.action.ActionForm, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)' has been removed	**protected**	N	1:1	Move; Add contextual data	Deleted, wasn't deprecated. New method is deprecated, delegates to processForwardConfig(). Release notes indicate that ActionForward is a subclass of ForwardConfig, so the two calls are interchangeable. Found by looking at caller of this method in old, looking at same caller in new library, seeing that new caller is completely rewritten and instead has a call-chain, near the end of that call chain is the new type which contains all the enclosed methods. The final method in the call chain also contains calls to this and other methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processContent(HttpServletResponse)[**void**]	org.apache.struts.action.RequestProcessor.processContent(HttpServletRequest, HttpServletResponse)[**void**]	7002	Method 'protected void processContent(javax.servlet.http.HttpServletResponse)' has been removed	**protected**	N	1:1	Move; Add contextual data	Deleted, wasn't deprecated. Found by looking at caller of this method in old, looking at same caller in new library, seeing that new caller is completely rewritten and instead has a call-chain, near the end of that call chain is the new type which contains all the enclosed methods. The final method in the call chain also contains calls to this and other methods.
org.apache.struts.action.ActionServlet.processLocale(HttpServletRequest)[**void**]	org.apache.struts.action.RequestProcessor.processLocale(HttpServletRequest, HttpServletResponse)[**void**]	7002	Method 'protected void processLocale(javax.servlet.http.HttpServletRequest)' has been removed	**protected**	N	1:1	Move; Add contextual data	Deleted, wasn't deprecated. Found by looking at caller of this method in old, looking at same caller in new library, seeing that new caller is completely rewritten and instead has a call-chain, near the end of that call chain is the new type which contains all the enclosed methods. The final method in the call chain also contains calls to this and other methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.processNoCache(HttpServletResponse)[**void**]	org.apache.struts.action.RequestProcessor.processNoCache(HttpServletRequest, HttpServletResponse)[**void**]	7002	Method '**protected void** processNoCache(javax.servlet.http.HttpServletResponse)' has been removed	**protected**	N	1:1	Move; Add contextual data	Deleted, wasn't deprecated. Found by looking at caller of this method in old, looking at same caller in new library, seeing that new caller is completely rewritten and instead has a call-chain, near the end of that call chain is the new type which contains all the enclosed methods. The final method in the call chain also contains calls to this and other methods.
org.apache.struts.action.ActionServlet.processPopulate(ActionForm, ActionMapping, HttpServletRequest)[**void**]	org.apache.struts.action.RequestProcessor.processPopulate(HttpServletRequest, HttpServletResponse, ActionForm, ActionMapping)[**void**]	7002	Method '**protected void** processPopulate(org.apache.struts.action.ActionForm, \code{org.apache.struts.action.ActionMapping, javax.servlet.http.HttpServletRequest)' has been removed	**protected**	N	1:1	Move; Add contextual data	Deleted, wasn't deprecated. Not deleted. Deleted, wasn't deprecated. Found by looking at caller of this method in old, looking at same caller in new library, seeing that new caller is completely rewritten and instead has a call-chain, near the end of that call chain is the new type which contains all the enclosed methods. The final method in the call chain also contains calls to this and other methods.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.getBufferSize()[**int**]		7002	Method '**public int** getBufferSize()' has been removed	**public**	N	1:1	Move	Deleted, wasn't deprecated; Moved into this class. Found by tracing its usage in a method in the old library, and finding a direct replacement in the new.
org.apache.struts.action.ActionServlet.getFormBeanClass()[String]	org.apache.struts.config.impl.ModuleConfigImpl.findFormBeans()[FormBeanConfig[]]; \code{org.apache.struts.config.FormBeanConfig.getName()[String]	7002	Method 'public java.lang.String getFormBeanClass()' has been removed	**public**	N	1:n	Change Type	Field the getter was for was also deleted; class this referenced was also deleted. I found this by matching the field associated with this getter. Note that the original returned a single class name; the new returns an array of FormBeanConfigs, each of which may be a different type.
org.apache.struts.action.ActionServlet.getForwardClass()[String]	org.apache.struts.config.impl.ModuleConfigImpl.findForwardClass()[ForwardConfig[]]; \code{org.apache.struts.config.ForwardConfig.getName()[String]	7002	Method '**public** java.lang.String getForwardClass()' has been removed	**public**	N	1:n	Change Type	Field the getter was for was also deleted; class this referenced was also deleted. I found this by matching the field associated with this getter. Note that the original returned a single class name; the new returns an array of FormBeanConfigs, each of which may be a different type.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.getMappingClass()[String]	org.apache.struts.config.impl.ModuleConfigImpl.getActionMappingClass()[String]	7002	Method 'public java.lang.String getMappingClass()' has been removed	public	N	1:1	Move; Rename	Getter was to a deleted field, field had a string literal. Found by noticing the method's code comment references the ActionMapping class, which also has a collector class ActionMappings. The code comment in ActionMappings indicated it had been replaced in the new library version by the ModuleConfig interface, which led me to this method.
org.apache.struts.action.ActionServlet.getMaxFileSize()[String]		7002	Method 'public java.lang.String getMaxFileSize()' has been removed	public	N	1:1	Move	Field associated with getter was deleted. Wasn't deprecated. Moved into Config class.
org.apache.struts.action.ActionServlet.getMultipartClass()[String]	org.apache.struts.config.ControllerConfig.getMultipartClass()[String]	7002	Method 'public java.lang.String getMultipartClass()' has been removed	public	N	1:1	Move	Field associated with getter was deleted. Wasn't deprecated. Moved into Config class.
org.apache.struts.action.ActionServlet.getTempDir()[String]	org.apache.struts.config.ControllerConfig.getTempDir()[String]	7002	Method 'public java.lang.String getTempDir()' has been removed	public	N	1:1	Move	Field associated with getter was deleted. Wasn't deprecated. Moved into Config class.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.addDataSource(String, DataSource)[**void**]	org.apache.struts.action.ActionServlet.initModuleDataSources(ModuleConfig config)[**void**]	7002	Method '**public void** addDataSource(java.lang.String, javax.sql.DataSource)' has been removed	**public**	N	1:n	Consolidate	Referenced by a nested, final class, that was deleted in the subsequent version. Wasn't deprecated. Found the replacement by looking at how the referenced field was used in the new library version. It looks like the library moved to having dataSources configured at startup, rather than added ad hoc.
org.apache.struts.action.ActionServlet.addFormBean(ActionFormBean)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.addFormBeanConfig(FormBeanConfig)[**void**]	7002	Method '**public void** addFormBean(org.apache.struts.action.ActionFormBean)' has been removed	**public**	N	1:n	Change Type	Deleted, wasn't deprecated. Only caller in library was also deleted. Found by looking at the code comment for the ActionFormBeans class in the new library, which indicated that the replacement would be here.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.addForward(ActionForward)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.addForwardConfig(ForwardConfig)[**void**] OR \code{org.apache.struts.config.ActionConfig.addForwardConfig(ForwardConfig)[**void**]	7002	Method '**public void** addForward(org.apache.struts.action.ActionForward)' has been removed	**public**	N	1:n	Change Type	Deleted, wasn't deprecated. Only caller in library was also deleted. Found by looking at the code comment for the ActionForwards class in the new library, which indicated that there were two replacement classes, but didn't offer any details as to when to use one over the other.
org.apache.struts.action.ActionServlet.addMapping(ActionMapping)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.addActionConfig(ActionConfig)[**void**]	7002	Method '**public void** addMapping(org.apache.struts.action.ActionMapping)' has been removed	**public**	N	1:n	Change Type	Deleted, wasn't deprecated. Only caller in library was also deleted. Found by looking at the code comment for the ActionMappings class in the new library, which indicated that the replacement would be here.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.reload()[**void**]	org.apache.struts.action.ActionServlet.destroy()[**void**]; \ code{org.apache.struts.action.ActionServlet.init()[**void**]	7002	Method '**public void** reload()' has been removed	**public**	N	1:n	Expose/Unpack Implementation	Deleted, wasn't deprecated. Essentially though this method was divided into two separate methods. FAQ indicates though that they expect that reload operations should be initiated on the container (e.g., on TOMCAT) rather then in code calls like this. That's potentially another replacement/transformation which could be difficult.
org.apache.struts.action.ActionServlet.removeFormBean(ActionFormBean)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.removeFormBeanConfig(FormBeanConfig)[**void**]	7002	Method '**public void** removeFormBean(org.apache.struts.action.ActionFormBean)' has been removed	**public**	N	1:n	Change Type	Deleted, wasn't deprecated. Only caller in library was also deleted. Found by looking at the code comment for the ActionFormBeans class in the new library, which indicated that the replacement would be here.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.removeForward(ActionForward)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.removeForwardConfig(ForwardConfig)[**void**]	7002	Method '**public void** removeForward(org.apache.struts.action.ActionForward)' has been removed	**public**	N	1:n	Change Type	Deleted, wasn't deprecated. Only caller in library was also deleted. Found by looking at the code comment for the ActionForwards class in the new library, which indicated that there were two replacement classes, but didn't offer any details as to when to use one over the other. However, ActionConfig has a state check which indicates that its meant to be configured once, and then frozen. There was no state check in the original code, nor is there one in ModuleConfigImpl, so I believe that's the best fit.
org.apache.struts.action.ActionServlet.removeMapping(ActionMapping)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.removeActionConfig(ActionConfig)[**void**]	7002	Method '**public void** removeMapping(org.apache.struts.action.ActionMapping)' has been removed	**public**	N	1:n	Change Type	Deleted, wasn't deprecated. Only caller in library was also deleted. Found by looking at the code comment for the ActionMappings class in the new library, which indicated that the replacement would be here.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.setBufferSize(**int**)[**void**]		7002	Method '**public void** setBufferSize(**int**)' has been removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into this class.
org.apache.struts.action.ActionServlet.setFormBeanClass(String)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.addFormBeanConfig(FormBeanConfig)[**void**]	7002	Method '**public void** setFormBeanClass(java.lang.String)' has been removed	**public**	N	1:n	Change Type	Deleted, wasn't deprecated. Only caller in library was also deleted. Found by looking at the code comment for the ActionFormBeans class in the new library, which indicated that the replacement would be here.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.setForwardClass(String)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.addForwardConfig(ForwardConfig)[**void**]	7002	Method '**public void** setForwardClass(java.lang.String)' has been removed	**public**	N	1:n	Change Type	Deleted, wasn't deprecated. Only caller in library was also deleted. Found by looking at the code comment for the ActionForwards class in the new library, which indicated that there were two replacement classes, but didn't offer any details as to when to use one over the other. However, ActionConfig has a state check which indicates that its meant to be configured once, and then frozen. There was no state check in the original code, nor is there one in ModuleConfigImpl, so I believe that's the best fit.
org.apache.struts.action.ActionServlet.setMappingClass(String)[**void**]	org.apache.struts.config.impl.ModuleConfigImpl.setActionMappingClass(String)[**void**]	7002	Method '**public void** setMappingClass(java.lang.String)' has been removed	**public**	N	1:1	Move; Rename	Deleted, wasn't deprecated. Only caller in library was also deleted. Found by looking at the code comment for the ActionMappings class in the new library, which indicated that the replacement would be here.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.setMaxFileSize(String)[**void**]		7002	Method '**public void** setMaxFileSize(java.lang.String)' has been removed	**public**	N	1:1	Move	Deleted, wasn't deprecated; Moved into this class.
org.apache.struts.action.ActionServlet.setMultipartClass(String)[**void**]	org.apache.struts.config.ActionConfig.setMultipartClass(String)[**void**]	7002	Method '**public void** setMultipartClass(java.lang.String)' has been removed	**public**	N	1:1	Move	Deleted, wasn't deprecated; Moved into this class.
org.apache.struts.action.ActionServlet.setTempDir(String)[**void**]	org.apache.struts.config.ControllerConfig.setTempDir(String)[**void**]	7002	Method '**public void** setTempDir(java.lang.String)' has been removed	**public**	N	1:1	Move	Deleted, wasn't deprecated; Moved into this class.
org.apache.struts.action.ActionServletWrapper.getMultipartClass()[String]	javax.servlet.ServletRequest.getAttribute(Globals.MULTIPART_KEY)[String]	7002	Method '**public** java.lang.String getMultipartClass()' has been removed	**public**	N	1:1	Expose/Unpack Implementation	Deleted, wasn't deprecated; The data was moved into a Servlet attribute.
org.apache.struts.actions.AddFormBeanAction	org.apache.struts.config.impl.ModuleConfigImpl.addFormBeanConfig(FormBeanConfig)[**void**]	8001	Class org.apache.struts.actions.AddFormBeanAction removed	**public**	N	1:n	Remap; Add contextual data	Deleted, wasn't deprecated. The class was a wrapper for an HttpServletRequest so that a method on the ActionServlet class could be invoked over HTTP; that method is deleted, and so this class is gone.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.actions.AddForwardAction	org.apache.struts.config.impl.ModuleConfigImpl.addForwardConfig(ForwardConfig)[**void**] OR \code{org.apache.struts.config.ActionConfig.addForwardConfig(ForwardConfig)[**void**]	8001	Class org.apache.struts.actions.AddForwardAction removed	**public**	N	1:n	Remap; Add contextual data	Deleted, wasn't deprecated. The class was a wrapper for an HttpServletRequest so that a method on the ActionServlet class could be invoked over HTTP; that method is deleted, and so this class is gone.
org.apache.struts.actions.AddMappingAction	org.apache.struts.config.impl.ModuleConfigImpl.addActionConfig(ActionConfig)[**void**]	8001	Class org.apache.struts.actions.AddMappingAction removed	**public**	N	1:n	Remap; Add contextual data	Deleted, wasn't deprecated. The class was a wrapper for an HttpServletRequest so that a method on the ActionServlet class could be invoked over HTTP; that method is deleted, and so this class is gone.
org.apache.struts.actions.ReloadAction	org.apache.struts.action.ActionServlet.destroy()[**void**];\code{org.apache.struts.action.ActionServlet.init()[**void**]	8001	Class org.apache.struts.actions.ReloadAction removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated. The class was a wrapper for an HttpServletRequest so that a method on the ActionServlet class could be invoked over HTTP; that method is deleted, and so this class is gone.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.actions.RemoveFormBeanAction	org.apache.struts.config.impl.ModuleConfigImpl.removeFormBeanConfig(FormBeanConfig)[**void**]	8001	Class org.apache.struts.actions.RemoveFormBeanAction removed	**public**	N	1:n	Remap; Add contextual data	Deleted, wasn't deprecated. The class was a wrapper for an HttpServletRequest so that a method on the ActionServlet class could be invoked over HTTP; that method is deleted, and so this class is gone.
org.apache.struts.actions.RemoveForwardAction	org.apache.struts.config.impl.ModuleConfigImpl.removeForwardConfig(ForwardConfig)[**void**]	8001	Class org.apache.struts.actions.RemoveForwardAction removed	**public**	N	1:n	Remap; Add contextual data	Deleted, wasn't deprecated. The class was a wrapper for an HttpServletRequest so that a method on the ActionServlet class could be invoked over HTTP; that method is deleted, and so this class is gone.
org.apache.struts.actions.RemoveMappingAction	org.apache.struts.config.impl.ModuleConfigImpl.removeActionConfig(ActionConfig)[**void**]	8001	Class org.apache.struts.actions.RemoveMappingAction removed	**public**	N	1:n	Remap; Add contextual data	Deleted, wasn't deprecated. The class was a wrapper for an HttpServletRequest so that a method on the ActionServlet class could be invoked over HTTP; that method is deleted, and so this class is gone.
org.apache.struts.digester.CallMethodRule	org.apache.commons.digester.CallMethodRule	8001	Class org.apache.struts.digester.CallMethodRule removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into an external library, apache.commons.digester

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.digester.CallParamRule	org.apache.commons.digester.CallParamRule	8001	Class org.apache.struts.digester.CallParamRule removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into an external library, apache.commons.digester
org.apache.struts.digester.Digester	org.apache.commons.digester.Digester	8001	Class org.apache.struts.digester.Digester removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into an external library, apache.commons.digester
org.apache.struts.digester.ObjectCreateRule	org.apache.commons.digester.ObjectCreateRule	8001	Class org.apache.struts.digester.ObjectCreateRule removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into an external library, apache.commons.digester
org.apache.struts.digester.Rule	org.apache.commons.digester.Rule	8001	Class org.apache.struts.digester.Rule removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into an external library, apache.commons.digester
org.apache.struts.digester.SetNextRule	org.apache.commons.digester.SetNextRule	8001	Class org.apache.struts.digester.SetNextRule removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into an external library, apache.commons.digester
org.apache.struts.digester.SetPropertiesRule	org.apache.commons.digester.SetPropertiesRule	8001	Class org.apache.struts.digester.SetPropertiesRule removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into an external library, apache.commons.digester
org.apache.struts.digester.SetPropertyRule	org.apache.commons.digester.SetPropertyRule	8001	Class org.apache.struts.digester.SetPropertyRule removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into an external library, apache.commons.digester

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.digester.SetTopRule	org.apache.commons.digester.SetTopRule	8001	Class org.apache.struts.digester.SetTopRule removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Moved into an external library, apache.commons.digester
org.apache.struts.taglib.BaseAttributeTag	org.apache.struts.taglib.logic.ConditionalTagBase	8001	Class org.apache.struts.taglib.BaseAttributeTag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated.
org.apache.struts.taglib.BaseFieldTag	org.apache.struts.taglib.html.BaseFieldTag	8001	Class org.apache.struts.taglib.BaseFieldTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.BaseHandlerTag	org.apache.struts.taglib.html.BaseHandlerTag	8001	Class org.apache.struts.taglib.BaseHandlerTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.BaseInputTag	org.apache.struts.taglib.html.BaseInputTag	8001	Class org.apache.struts.taglib.BaseInputTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.BaseTag	org.apache.struts.taglib.html.BaseTag	8001	Class org.apache.struts.taglib.BaseTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.ButtonTag	org.apache.struts.taglib.html.ButtonTag	8001	Class org.apache.struts.taglib.ButtonTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.CancelTag	org.apache.struts.taglib.html.CancelTag	8001	Class org.apache.struts.taglib.CancelTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.CheckboxTag	org.apache.struts.taglib.html.CheckboxTag	8001	Class org.apache.struts.taglib.CheckboxTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.Constants	org.apache.struts.taglib.html.Constants	8001	Class org.apache.struts.taglib.Constants removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.EncodeRedirectURLTag	org.apache.struts.taglib.logic.ForwardTag	8001	Class org.apache.struts.taglib.EncodeRedirectURLTag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated. The doEndTag() has the relevant functionality.
org.apache.struts.taglib.EncodeURLTag	org.apache.struts.util.RequestUtils.encodeURL(String)[String]	8001	Class org.apache.struts.taglib.EncodeURLTag removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Deleted, wasn't deprecated.
org.apache.struts.taglib.EnumerateTag	org.apache.struts.taglib.logic.IterateTag	8001	Class org.apache.struts.taglib.EnumerateTag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated.
org.apache.struts.taglib.EnumerateTei	org.apache.struts.taglib.logic.IterateTei	8001	Class org.apache.struts.taglib.EnumerateTei removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.ErrorsTag	org.apache.struts.taglib.html.ErrorsTag	8001	Class org.apache.struts.taglib.ErrorsTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.FileTag	org.apache.struts.taglib.html.FileTag	8001	Class org.apache.struts.taglib.FileTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.FormTag	org.apache.struts.taglib.html.FormTag	8001	Class org.apache.struts.taglib.FormTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.ForwardTag	org.apache.struts.taglib.logic.ForwardTag	8001	Class org.apache.struts.taglib.ForwardTag removed	**public**	N	1:1	Move	Deleted, wasn't deprecated. Note that the replacement already existed in the previous version, and there's some slight differences in functionality.
org.apache.struts.taglib.GetPropertyTag	org.apache.struts.taglib.bean.DefineTag	8001	Class org.apache.struts.taglib.GetPropertyTag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated.
org.apache.struts.taglib.GetPropertyTei	org.apache.struts.taglib.bean.DefineTei	8001	Class org.apache.struts.taglib.GetPropertyTei removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated.
org.apache.struts.taglib.HiddenTag	org.apache.struts.taglib.html.HiddenTag	8001	Class org.apache.struts.taglib.HiddenTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.HtmlPropertyTag	org.apache.struts.taglib.bean.WriteTag	8001	Class org.apache.struts.taglib.HtmlPropertyTag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated.
org.apache.struts.taglib.IfAttributeExistsTag	org.apache.struts.taglib.logic.ConditionalTagBase	8001	Class org.apache.struts.taglib.IfAttributeExistsTag removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Subclass of BaseAttributeTag, which was deleted. Not deprecated prior. Replacement is to subclass this class, and basically implement the simple condition test by overriding the one relevant method.
org.apache.struts.taglib.IfAttributeMissingTag	org.apache.struts.taglib.logic.ConditionalTagBase	8001	Class org.apache.struts.taglib.IfAttributeMissingTag removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Subclass of BaseAttributeTag, which was deleted. Not deprecated prior. Replacement is to subclass this class, and basically implement the simple condition test by overriding the one relevant method.
org.apache.struts.taglib.IfParameterEqualsTag	org.apache.struts.taglib.logic.ConditionalTagBase	8001	Class org.apache.struts.taglib.IfParameterEqualsTag removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Subclass of BaseAttributeTag, which was deleted. Not deprecated prior. Replacement is to subclass this class, and basically implement the simple condition test by overriding the one relevant method.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.IfParameterNotEqualsTag	org.apache.struts.taglib.logic.ConditionalTagBase	8001	Class org.apache.struts.taglib.IfParameterNotEqualsTag removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Subclass of BaseAttributeTag, which was deleted. Not deprecated prior. Replacement is to subclass this class, and basically implement the simple condition test by overriding the one relevant method.
org.apache.struts.taglib.IfParameterNotNullTag	org.apache.struts.taglib.logic.ConditionalTagBase	8001	Class org.apache.struts.taglib.IfParameterNotNullTag removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Subclass of BaseAttributeTag, which was deleted. Not deprecated prior. Replacement is to subclass this class, and basically implement the simple condition test by overriding the one relevant method.
org.apache.struts.taglib.IfParameterNullTag	org.apache.struts.taglib.logic.ConditionalTagBase	8001	Class org.apache.struts.taglib.IfParameterNullTag removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Subclass of BaseAttributeTag, which was deleted. Not deprecated prior. Replacement is to subclass this class, and basically implement the simple condition test by overriding the one relevant method.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.IfPropertyEqualsTag	org.apache.struts.taglib.logic.ConditionalTagBase	8001	Class org.apache.struts.taglib.IfPropertyEqualsTag removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Subclass of BaseAttributeTag, which was deleted. Not deprecated prior. Replacement is to subclass this class, and basically implement the simple condition test by overriding the one relevant method.
org.apache.struts.taglib.IfPropertyNotEqualsTag	org.apache.struts.taglib.logic.ConditionalTagBase	8001	Class org.apache.struts.taglib.IfPropertyNotEqualsTag removed	**public**	N	1:n	Expose/Unpack Implementation; Remap	Subclass of BaseAttributeTag, which was deleted. Not deprecated prior. Replacement is to subclass this class, and basically implement the simple condition test by overriding the one relevant method.
org.apache.struts.taglib.IncludeTag	org.apache.struts.taglib.bean.IncludeTag	8001	Class org.apache.struts.taglib.IncludeTag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated.
org.apache.struts.taglib.IterateTag	org.apache.struts.taglib.logic.IterateTag	8001	Class org.apache.struts.taglib.IterateTag removed	**public**	N	Delete	Delete	Deleted, wasn't deprecated.
org.apache.struts.taglib.IterateTei	org.apache.struts.logic.taglib.IterateTei	8001	Class org.apache.struts.taglib.IterateTei removed	**public**	N	Delete	Delete	Deleted, wasn't deprecated.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.Link1Tag	org.apache.struts.taglib.html.LinkTag	8001	Class org.apache.struts.taglib.Link1Tag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated. The old version was meant to provide JDK 1.1 support, this new version requires JDK 1.2
org.apache.struts.taglib.LinkTag	org.apache.struts.taglib.html.LinkTag	8001	Class org.apache.struts.taglib.LinkTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.MessageTag	org.apache.struts.taglib.bean.MessageTag	8001	Class org.apache.struts.taglib.MessageTag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated.
org.apache.struts.taglib.MultiboxTag	org.apache.struts.taglib.html.MultiboxTag	8001	Class org.apache.struts.taglib.MultiboxTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.OptionTag	org.apache.struts.taglib.html.OptionTag	8001	Class org.apache.struts.taglib.OptionTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.Options1Tag	org.apache.struts.taglib.html.OptionsTag	8001	Class org.apache.struts.taglib.Options1Tag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated. The old version was meant to provide JDK 1.1 support, this new version requires JDK 1.2
org.apache.struts.taglib.OptionsTag	org.apache.struts.taglib.html.OptionsTag	8001	Class org.apache.struts.taglib.OptionsTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.ParameterTag	org.apache.struts.taglib.bean.ParameterTag	8001	Class org.apache.struts.taglib.ParameterTag removed	**public**	N	1:n	Remap	Deleted, wasn't deprecated.
org.apache.struts.taglib.PasswordTag	org.apache.struts.taglib.html.PasswordTag	8001	Class org.apache.struts.taglib.PasswordTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.PropertyTag	org.apache.struts.taglib.bean.WriteTag	8001	Class org.apache.struts.taglib.PropertyTag removed	**public**	N	Delete	Delete	Deleted, wasn't deprecated.
org.apache.struts.taglib.RadioTag	org.apache.struts.taglib.html.RadioTag	8001	Class org.apache.struts.taglib.RadioTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.RedirectTag	org.apache.struts.taglib.logic.RedirectTag	8001	Class org.apache.struts.taglib.RedirectTag removed	**public**	N	1:n	Change Type	Deleted, wasn't deprecated. This replacement has the same name and similar methods, but works slightly differently than the original, and so can't be considered as a Move refactoring. The developer will need to wrap some additional code around this to make it work like the original.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.ResetTag	org.apache.struts.taglib.html.ResetTag	8001	Class org.apache.struts.taglib.ResetTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.SelectTag	org.apache.struts.taglib.html.SelectTag	8001	Class org.apache.struts.taglib.SelectTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.SubmitTag	org.apache.struts.taglib.html.SubmitTag	8001	Class org.apache.struts.taglib.SubmitTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.TextTag	org.apache.struts.taglib.html.TextTag	8001	Class org.apache.struts.taglib.TextTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.TextareaTag	org.apache.struts.taglib.html.TextareaTag	8001	Class org.apache.struts.taglib.TextareaTag removed	**public**	N	1:1	Move	Moved into another package. Documented in release notes, but not deprecated/linked prior.
org.apache.struts.taglib.html.FormTag.getActionMappingName()[String]	org.apache.struts.util.RequestUtils.getActionMappingName(String)[String]	7002	Method '**protected** java.lang.String getActionMappingName()' has been removed	**protected**	N	1:1	Add contextual data	Deleted, wasn't deprecated. Functionality was extracted into an external static method. Found by searching on the associated code comment.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.html.FormTag.getActionMappingURL()[String]	org.apache.struts.util.RequestUtils.getActionMappingURL(String, PageContext)[String]	7002	Method '**protected** java.lang.String getActionMappingURL()' has been removed	**protected**	N	1:1	Add contextual data	Deleted, wasn't deprecated. Functionality was extracted into an external static method. Found by searching on the associated code comment.
org.apache.struts.taglib.html.ImageTag:alt[String]	org.apache.struts.taglib.html.BaseHandlerTag:getAlt()[String]; org.apache.struts.taglib.html.BaseHandlerTag:setAlt(String)[**void**]	6001	Removed field alt	**protected**	N	1:n	Pull Up; Hide; Encapsulate	Deleted, wasn't deprecated. Moved up into a superclass, changed visibility to **private**, and encapsulated with getters/setters.
org.apache.struts.taglib.html.ImageTag:altKey[String]	org.apache.struts.taglib.html.BaseHandlerTag:getAltKey()[String]; org.apache.struts.taglib.html.BaseHandlerTag:setAltKey(String)[**void**]	6001	Removed field altKey	**protected**	N	1:n	Pull Up; Hide; Encapsulate	Deleted, wasn't deprecated. Moved up into a superclass, changed visibility to **private**, and encapsulated with getters/setters.
org.apache.struts.taglib.html.ImageTag:bundle[String]	org.apache.struts.taglib.html.BaseHandlerTag:getBundle()[String]; org.apache.struts.taglib.html.BaseHandlerTag:setBundle(String)[**void**]	6001	Removed field bundle	**protected**	N	1:n	Pull Up; Hide; Encapsulate	Deleted, wasn't deprecated. Moved up into a superclass, changed visibility to **private**, and encapsulated with getters/setters.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.html.ImageTag:defaultLocale[Locale]	org.apache.struts.taglib.html.BaseHandlerTag:defaultLocale[Locale]	6001	Removed field defaultLocale	**protected**	N	1:1	Pull Up	Deleted, wasn't deprecated. Moved up into a superclass.
org.apache.struts.taglib.html.ImageTag:locale[String]	org.apache.struts.taglib.html.BaseHandlerTag:getLocale()[String]; org.apache.struts.taglib.html.BaseHandlerTag:setLocale(String)[**void**]	6001	Removed field locale	**protected**	N	1:n	Pull Up; Hide; Encapsulate	Deleted, wasn't deprecated. Moved up into a superclass, changed visibility to **private**, and encapsulated with getters/setters.
org.apache.struts.taglib.html.ResetTag:name[String]	org.apache.struts.taglib.html.ResetTag:property[String]	6001	Removed field name	**protected**	N	1:1	Rename	Code comment associated with the fields are unchanged between versions.
org.apache.struts.taglib.html.ResetTag.getName()[String]	org.apache.struts.taglib.html.ResetTag.getProperty()[String]	7002	Method '**public** java.lang.String getName()' has been removed	**public**	N	1:1	Rename	Code comment associated with the methods are unchanged between versions.
org.apache.struts.taglib.html.ResetTag.setName(String)[**void**]	org.apache.struts.taglib.html.ResetTag.setProperty(String)[**void**]	7002	Method '**public void** setName(java.lang.String)' has been removed	**public**	N	1:1	Rename	Code comment associated with the methods are unchanged between versions.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.upload.DiskMultipartRequestHandler.getMaxSizeFromServlet()[**long**]	org.apache.struts.config.ControllerConfig.getMaxFileSize()[String]; org.apache.struts.upload.DiskMultipartRequestHandler.getMaxFileSize(String)[**long**]	7002	Method '**protected long** getMaxSizeFromServlet()' has been removed	**protected**	N	1:n	Expose/Unpack Implementation	Deleted, wasn't deprecated. This is a 1:n replacement; the replacement is a method which wraps a call to another method.
org.apache.struts.upload.DiskMultipartRequestHandler.retrieveTempDir()[**void**]	org.apache.struts.upload.DiskMultipartRequestHandler.retrieveTempDir(ModuleConfig)[**void**]	7004	In method '**protected void** retrieveTempDir()' the number of arguments has changed	**protected**	N	1:1	Add contextual data	Added a parameter.
org.apache.struts.upload.MultipartIterator:MAX_LINE_SIZE[**int**]	org.apache.struts.upload.MultipartBoundaryInputStream:DEFAULT_LINE_SIZE[**int**]	6001	Removed field MAX_LINE_SIZE	**public**	N	1:1	Move; Hide	Deleted, wasn't deprecated. This looks like code-cleanup on something that maybe should not have been in this state in an API release. Was publicly accessible, and **static** (yet not **final**), so anyone could change it. Now, there's no way to get the value.
org.apache.struts.upload.MultipartIterator:boundaryBytes[**byte**[]]	org.apache.struts.upload.MultipartBoundaryInputStream:boundaryBytes[**byte**[]]	6001	Removed field boundaryBytes	**protected**	N	1:1	Move; Encapsulate(Partial)	Deleted, wasn't deprecated. This looks like code-cleanup on something that maybe should not have been in this state in an API release.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.upload.MultipartIterator:contentRead[**boolean**]	org.apache.struts.upload.MultipartBoundaryInputStream:endOfStream[**boolean**]	6001	Removed field contentRead	**protected**	N	1:1	Move; Encapsulate(Partial)	Deleted, wasn't deprecated. Found the replacement after noticing the change in type in the inputStream field, and looking to see if maybe it had an equivalent functionality. While it's a direct replacement, because it's a **protected** field on a type stored locally in the class, getters would need to be called.
org.apache.struts.upload.MultipartIterator:inputStream[BufferedMultipartInputStream]	org.apache.struts.upload.MultipartBoundaryInputStream:inputStream[MultipartBoundaryInputStream]	6004	Changed type of field inputStream from org.apache.struts.upload.BufferedMultipartInputStream to \code{org.apache.struts.upload.MultipartBoundaryInputStream	**protected**	N	1:n	Change Type	Type change on field. The new type extends the same superclass, but has a very different method/field list, and would be a substantial rework to implement.
org.apache.struts.upload.MultipartIterator:totalLength[**long**]	org.apache.struts.upload.MultipartBoundaryInputStream:bytesRead[**long**]	6001	Removed field totalLength	**protected**	N	1:n	Move; Encapsulate(Partial)	Deleted, wasn't deprecated. This looks like code-cleanup on something that maybe should not have been in this state in an API release. Major cleanup of File I/O meant this wasn't needed anymore.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.upload.MultipartIterator.readLine()[String]	org.apache.struts.upload.MultipartBoundaryInputStream.readLine()[String]	7002	Method '**protected** java.lang.String readLine()' has been removed	**protected**	N	1:1	Move; Hide	Deleted, wasn't deprecated. This looks like code-cleanup on something that maybe should not have been in this state in an API release. The new method is heavily rewritten and **private** now, so there's now way to get access to it.
org.apache.struts.upload.MultipartIterator.equals(**byte**[], **int**, **int**, **byte**[])[**boolean**]	org.apache.struts.upload.MultipartBoundaryInputStream.equals(**byte**[], **int**, **int**, **byte**[])[**boolean**]	7002	Method '**public boolean** equals(**byte**[], **int**, **int**, **byte**[])' has been removed	**public**	N	1:1	Hide	Deleted, wasn't deprecated. This looks like code-cleanup on something that maybe should not have been in this state in an API release. Major cleanup of File I/O meant this wasn't needed anymore.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.upload.MultipartIterator.parseBoundary(String)[String]	org.apache.struts.upload.MultipartIterator.getBoundaryFromContentyType()[**void**]	7002	Method '**public** java.lang.String parseBoundary(java.lang.String)' has been removed	**public**	N	1:1	Hide	Rewritten and made **private**. There is replacement functionality, but the developers clearly don't want this piece of code publicly accessed as it seems very implementation specific and likely to change, so they rewrote it, renamed it, and made it **private** AND **final**, so it can't be reused at all.
org.apache.struts.upload.MultipartIterator.parseContentType(String)[String]	org.apache.struts.upload.MultipartIterator.getContentTypeOfRequest()[**void**]	7002	Method '**public** java.lang.String parseContentType(java.lang.String)' has been removed	**public**	N	1:1	Hide	Rewritten and made **private**. There is replacement functionality, but the developers clearly don't want this piece of code publicly accessed as it seems very implementation specific and likely to change, so they rewrote it, renamed it, and made it **private** AND **final**, so it can't be reused at all.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.upload.MultipartIterator.parseDispositionFilename(String)[String]	org.apache.struts.upload.MultipartBoundaryInputStream.parseForParameter(String, String)[String]	7002	Method '**public** java.lang.String parseDispositionFilename(java.lang.String)' has been removed	**public**	N	1:1	Move; Hide	Deleted, wasn't deprecated. This looks like code-cleanup on something that maybe should not have been in this state in an API release. Major cleanup of File I/O meant this wasn't needed anymore.
org.apache.struts.upload.MultipartIterator.parseDispositionName(String)[String]	org.apache.struts.upload.MultipartBoundaryInputStream.parseForParameter(String, String)[String]	7002	Method '**public** java.lang.String parseDispositionName(java.lang.String)' has been removed	**public**	N	1:1	Move; Hide	Deleted, wasn't deprecated. This looks like code-cleanup on something that maybe should not have been in this state in an API release. Major cleanup of File I/O meant this wasn't needed anymore.
org.apache.struts.upload.MultipartIterator.parseForAttribute(String, String)[String]	org.apache.struts.upload.MultipartBoundaryInputStream.parseForParameter(String, String)[String]	7002	Method '**public** java.lang.String parseForAttribute(java.lang.String, java.lang.String)' has been removed	**public**	N	1:1	Move; Hide	Deleted, wasn't deprecated. This looks like code-cleanup on something that maybe should not have been in this state in an API release. Major cleanup of File I/O meant this wasn't needed anymore.
org.apache.struts.util.ArrayStack	org.apache.commons.collections.ArrayStack	8001	Class org.apache.struts.util.ArrayStack removed	**public**	N	1:1	Extract to external API	Deleted, wasn't deprecated. Essentially moved into a separate library.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.BeanUtils	org.apache.commons.beanutils.BeanUtils	8001	Class org.apache.struts.util.BeanUtils removed	**public**	Y	1:1	Extract to external API	Deprecated before deleted, although replacement class wasn't specified. Essentially moved into a separate library.
org.apache.struts.util.ConvertUtils	org.apache.commons.beanutils.ConvertUtils	8001	Class org.apache.struts.util.ConvertUtils removed	**public**	Y	1:1	Extract to external API	Deprecated before deleted, although replacement class wasn't specified. Essentially moved into a separate library.
org.apache.struts.util.FastArrayList	org.apache.commons.collections.FastArrayList	8001	Class org.apache.struts.util.FastArrayList removed	**public**	Y	1:1	Extract to external API	Deprecated before deleted, although replacement class wasn't specified. Essentially moved into a separate library.
org.apache.struts.util.FastHashMap	org.apache.commons.collections.FastHashMap	8001	Class org.apache.struts.util.FastHashMap removed	**public**	Y	1:1	Extract to external API	Deprecated before deleted, although replacement class wasn't specified. Essentially moved into a separate library.
org.apache.struts.util.FastTreeMap	org.apache.commons.collections.FastTreeMap	8001	Class org.apache.struts.util.FastTreeMap removed	**public**	Y	1:1	Extract to external API	Deprecated before deleted, although replacement class wasn't specified. Essentially moved into a separate library.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.GenericConnection	org.apache.struts.legacy.GenericConnection	8001	Class org.apache.struts.util.GenericConnection removed	**public**	N	1:1	Extract to external API	Deleted, wasn't deprecated. Moved into a legacy library to support backwards compatibility issues with DB's.
org.apache.struts.util.GenericDataSource:activeCount[**int**]	org.apache.struts.legacy.GenericDataSource:activeCount[unknown]	6001	Removed field activeCount	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:autoCommit[**boolean**]	org.apache.struts.legacy.GenericDataSource:autoCommit[**boolean**]	6001	Removed field autoCommit	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:closed[**boolean**]	org.apache.struts.legacy.GenericDataSource:closed[**boolean**]	6001	Removed field closed	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.GenericDataSource:connections[LinkedList]	org.apache.struts.legacy.GenericDataSource:connections[LinkedList]	6001	Removed field connections	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:debug[**int**]	org.apache.struts.legacy.GenericDataSource:debug[**int**]	6001	Removed field debug	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:description[String]	org.apache.struts.legacy.GenericDataSource:description[String]	6001	Removed field description	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:driver[Driver]	org.apache.struts.legacy.GenericDataSource:driver[Driver]	6001	Removed field driver	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.GenericDataSource:driverClass[String]	org.apache.struts.legacy.GenericDataSource:driverClass[String]	6001	Removed field driverClass	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:logWriter[PrintWriter]	org.apache.struts.legacy.GenericDataSource:logWriter[PrintWriter]	6001	Removed field logWriter	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:loginTimeout[**int**]	org.apache.struts.legacy.GenericDataSource:loginTimeout[**int**]	6001	Removed field loginTimeout	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:maxCount[**int**]	org.apache.struts.legacy.GenericDataSource:maxCount[**int**]	6001	Removed field maxCount	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.GenericDataSource:minCount[int]	org.apache.struts.legacy.GenericDataSource:minCount[int]	6001	Removed field minCount	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:password[String]	org.apache.struts.legacy.GenericDataSource:password[String]	6001	Removed field password	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:pingCommand[String]	org.apache.struts.legacy.GenericDataSource:pingCommand[String]	6001	Removed field pingCommand	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:pingQuery[String]	org.apache.struts.legacy.GenericDataSource:pingQuery[String]	6001	Removed field pingQuery	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.GenericDataSource:properties[Properties]	org.apache.struts.legacy.GenericDataSource:properties[Properties]	6001	Removed field properties	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:readOnly[**boolean**]	org.apache.struts.legacy.GenericDataSource:readOnly[**boolean**]	6001	Removed field readOnly	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:url[String]	org.apache.struts.legacy.GenericDataSource:url[String]	6001	Removed field url	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.GenericDataSource:useCount[**int**]	org.apache.struts.legacy.GenericDataSource:useCount[**int**]	6001	Removed field useCount	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.

Continued on next page

Table E.4 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.GenericDataSource:user[String]	org.apache.struts.legacy.GenericDataSource:user[String]	6001	Removed field user	**protected**	N	1:1	Extract to external API	Deleted, wasn't deprecated. In the new version, the new class is deprecated, but extends this legacy class which contains all of the fields.
org.apache.struts.util.PropertyUtils	org.apache.commons.beanutils.PropertyUtils	8001	Class org.apache.struts.util.PropertyUtils removed	**public**	Y	1:1	Move	Deprecated before deleted, although replacement class wasn't specified. Essentially moved into a separate library.

Table E.5: APACHE STRUTS 1.1.0 to 1.2.4 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.Action. toHex(**byte**[])[String]	org.apache.struts.util. TokenProcessor.toHex(**byte**[])[String]	7002	Method '**protected** java.lang. String toHex(**byte**[])' has been removed	**protected**	Y	1:n	Move; Hide	Was deprecated before being deleted, no replacement offered but it was just delegated to this method.
org.apache.struts.action.Action. getResources()[MessageResources]	org.apache.struts.action.Action. getResources(HttpServletRequest)[MessageResources]	7002	Method '**protected** org.apache. struts.util.MessageResources getResources()' has been removed	**protected**	Y	1:n	Add contextual data	Was deprecated, replacement described in comments.
org.apache.struts.action.Action. perform(ActionMapping, ActionForm, ServletRequest, ServletResponse)[ActionForward]	org.apache.struts.action.Action. execute(ActionMapping, ActionForm, ServletRequest, ServletResponse)[ActionForward]	7002	Method '**public** org.apache. struts.action.ActionForward perform(org.apache.struts.action. ActionMapping, org.apache.struts .action.ActionForm, javax.servlet. ServletRequest, javax.servlet. ServletResponse)' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement described in comments.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.Action. perform(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)[ActionForward]	org.apache.struts.action.Action. execute(ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)[ActionForward]	7002	Method '**public** org.apache. struts.action.ActionForward perform(org.apache.struts.action. ActionMapping, org.apache.struts .action.ActionForm, javax.servlet. http.HttpServletRequest, javax. servlet.http.HttpServletResponse) ' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement described in comments.
org.apache.struts.action. ActionException	org.apache.struts.config. ExceptionConfig	8001	Class org.apache.struts.action. ActionException removed	**public**	Y	1:1	Shorten subtype chain	Was deprecated, replacement described in comments.
org.apache.struts.action. ActionFormBeans	org.apache.struts.config. ModuleConfig; org.apache.struts.config. FormBeanConfig	8001	Class org.apache.struts.action. ActionFormBeans removed	**public**	Y	1:n	Change Type	Was deprecated, replacement partially described in comments.
org.apache.struts.action. ActionForwards	org.apache.struts.config. ModuleConfig; org.apache.struts.config. ForwardConfig	8001	Class org.apache.struts.action. ActionForwards removed	**public**	Y	1:n	Change Type	Was deprecated, replacement partially described in comments.
org.apache.struts.action. ActionMappings	org.apache.struts.config. ModuleConfig; org.apache.struts.config. ActionConfig	8001	Class org.apache.struts.action. ActionMappings removed	**public**	Y	1:n	Change Type	Was deprecated, replacement partially described in comments.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionMessages.empty()[**boolean**]	org.apache.struts.action.ActionMessages.isEmpty()[**boolean**]	7002	Method '**public boolean** empty()' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement described in comments.
org.apache.struts.action.ActionMessages$ActionMessageItem.ActionMessageItem(List, **int**)	org.apache.struts.action.ActionMessages$ActionMessageItem.ActionMessageItem(List, **int**, String)	7004	In method '**public** ActionMessages$ActionMessageItem(org.apache.struts.action.ActionMessages, java.util.List, **int**)' the number of arguments has changed	**public**	N	1:1	Add contextual data	Additional parameter added to a nested class's constructor
org.apache.struts.action.ActionServlet:debug[**int**]	none	6001	Removed field debug	**protected**	Y	1:1	Extract to External API	Was deprecated, replacment partially described in comments, and release notes. Essentially, you're expected to either (a) utilize your own logging now, or (b) utilize the commons logging framework. There's some straightfoward replacments using LOG4J for this code.
org.apache.struts.action.ActionServlet.getApplicationConfig(HttpServletRequest)[ApplicationConfig]	org.apache.struts.action.ActionServlet.getModuleConfig(HttpServletRequest)[ModuleConfig]	7002	Method '**protected** org.apache.struts.config.ApplicationConfig getApplicationConfig(javax.servlet.http.HttpServletRequest)' has been removed	**protected**	Y	1:1	Shorten subtype chain	Was deprecated, replacement described in comments.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.initApplicationConfig(String, String)[ApplicationConfig]	org.apache.struts.action.ActionServlet.initModuleConfig(String, String)[ModuleConfig]	7002	Method '**protected** org.apache.struts.config.ApplicationConfig initApplicationConfig(java.lang.String, java.lang.String)' has been removed	**protected**	Y	1:1	Shorten subtype chain	Was deprecated, replacement described in comments.
org.apache.struts.action.ActionServlet.destroyApplications()[**void**]	org.apache.struts.action.ActionServlet.destroyModules()[**void**]	7002	Method '**protected void** destroyApplications()' has been removed	**protected**	Y	1:1	Rename	Was deprecated, replacement described in comments.
org.apache.struts.action.ActionServlet.destroyDataSources()[**void**]	none	7002	Method '**protected void** destroyDataSources()' has been removed	**protected**	Y	1:n	Extract to External API	Was deprecated, no replacement offered: comments just say this won't be required anymore. What's happening is that this method used to look after getting rid of JDBC connections. But now, users of the STRUTS API are required to manage their own JDBC and DB connections, either through the container or on their own. Because of this, the callers of the library will be looking after their own JDBC calls. So the replacement should be a call on a JDBC connector to close.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.initApplicationDataSources(ModuleConfig)[**void**]	org.apache.struts.action.ActionServlet.initModuleDataSources(ModuleConfig)[**void**]		Method '**protected void** initApplicationDataSources(org.apache.struts.config.ModuleConfig)' has been removed	**protected**	Y	1:1	Rename	Was deprecated, replacement described in comments. Note that this was a delegated method, and now the need to delegate is eliminated. Should that be classified as a Rename?
org.apache.struts.action.ActionServlet.initApplicationMessageResources(ModuleConfig)[**void**]	org.apache.struts.action.ActionServlet.initModuleMessageResources(ModuleConfig)[**void**]	7002	Method '**protected void** initApplicationMessageResources(org.apache.struts.config.ModuleConfig)' has been removed	**protected**	Y	1:1	Rename	Was deprecated, replacement described in comments.
org.apache.struts.action.ActionServlet.initApplicationPlugIns(ModuleConfig)[**void**]	org.apache.struts.action.ActionServlet.initModulePlugIns(ModuleConfig)[**void**]	7002	Method '**protected void** initApplicationPlugIns(org.apache.struts.config.ModuleConfig)' has been removed	**protected**	Y	1:1	Rename	Was deprecated, replacement described in comments.
org.apache.struts.action.ActionServlet.initDataSources()[**void**]	org.apache.struts.action.ActionServlet.initModuleDataSources(ModuleConfig)[**void**]	7002	Method '**protected void** initDataSources()' has been removed	**protected**	Y	1:n	Add contextual data; Rename	Was deprecated, code was damaged in the old version to do nothing. The replacement described in comments was also deprecated and removed in the next library version

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.getDebug()[**int**]	none	7002	Method '**public int** getDebug()' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacment partially described in comments, and release notes. Essentially, you're expected to either (a) utilize your own logging now, or (b) utilize the commons logging framework. There's some straightfoward replacments using LOG4J for this code.
org.apache.struts.action.ActionServlet.findDataSource(String)[DataSource]	org.apache.struts.config.ModuleConfig.findDataSourceConfigs(); org.apache.struts.util.RequestUtils.applicationInstance(DataSourceConfig.getType())	7002	Method '**public** javax.sql.DataSource findDataSource(java.lang.String)' has been removed	**public**	Y	1:n	Change Type; Expose/Unpack Implementation	Was deprecated, comments just say to get the DataSources from the servlet context. Replacement is divined from how the current code handles this. 1:n
org.apache.struts.action.ActionServlet.findFormBean(String)[ActionFormBean]	org.apache.struts.config.ModuleConfig.findFormBeanConfig(); org.apache.struts.util.RequestUtils.applicationInstance(FormBeanConfig.getType())	7002	Method '**public** org.apache.struts.action.ActionFormBean findFormBean(java.lang.String)' has been removed	**public**	Y	1:n	Change Type; Expose/Unpack Implementation	Was deprecated, replacement partially described in comments—replacement isn't the same type, and seems to be an intermediate step.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.findForward(String)[ActionForward]	org.apache.struts.config.ModuleConfig.findForwardConfig(); org.apache.struts.util.RequestUtils.applicationInstance(ForwardConfig.getType())	7002	Method '**public** org.apache.struts.action.ActionForward findForward(java.lang.String)' has been removed	**public**	Y	1:n	Change Type; Expose/Unpack Implementation	Was deprecated, replacement partially described in comments—replacement isn't the same type, and seems to be an intermediate step. Not sure what the next step should be. 1:n
org.apache.struts.action.ActionServlet.findMapping(String)[ActionMapping]	org.apache.struts.config.ModuleConfig.findActionConfig(); org.apache.struts.util.RequestUtils.applicationInstance(ActionConfig.getType())	7002	Method '**public** org.apache.struts.action.ActionMapping findMapping(java.lang.String)' has been removed	**public**	Y	1:n	Change Type; Expose/Unpack Implementation	Was deprecated, replacement partially described in comments—replacement isn't the same type, and seems to be an intermediate step. Not sure what the next step should be. 1:n
org.apache.struts.action.ActionServlet.getResources()[MessageResources]	org.apache.struts.action.Action.getResources(HttpServletRequest)	7002	Method '**public** org.apache.struts.util.MessageResources getResources()' has been removed	**public**	Y	1:n	Change Type	Was deprecated, replacement described in comments. Needs a parameter.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServlet.log(String, **int**)[**void**]	none	7002	Method '**public void** log(java.lang.String, **int**)' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacment partially described in comments, and release notes. Essentially, you're expected to either (a) utilize your own logging now, or (b) utilize the commons logging framework. There's some straightfoward replacments using LOG4J for this code.
org.apache.struts.action.ActionServletWrapper.log(String)[**void**]	none	7002	Method '**public void** log(java.lang.String)' has been removed	**public**	N	1:n	Extract to External API	Was deprecated, replacment partially described in comments, and release notes. Essentially, you're expected to either (a) utilize your own logging now, or (b) utilize the commons logging framework. There's some straightfoward replacments using LOG4J for this code.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.ActionServletWrapper.log(String, **int**)[**void**]	none	7002	Method '**public void** log(java.lang.String, **int**)' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacement partially described in comments, and release notes. Essentially, you're expected to either (a) utilize your own logging now, or (b) utilize the commons logging framework. There's some straightforward replacements using LOG4J for this code.
org.apache.struts.action.DynaActionFormClass: dynaClasses[FastHashMap]	org.apache.struts.config.FormBeanConfig:name[String]; org.apache.struts.config.FormBeanConfig: dynaActionFormClass[DynaActionFormClass]	6001	Removed field dynaClasses	**protected**	N	1:n	Change Type; Expose/Unpack Implementation	Wasn't deprecated, no replacement. The field was used to cache instances of this class in one location. Now, it looks like each FormBeanConfig class has its own instance of the DynaActionFormClass.
org.apache.struts.action.DynaActionFormClass:lock[String]	org.apache.struts.config.FormBeanConfig:lock[String]	6001	Removed field lock	**protected**	N	1:1	Move	Wasn't deprecated, no replacement. This field was used as a hack to ensure that they could synchronize an object (the earlier dynaClasses field) even if it was null at some points.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.action.RequestProcessor:appConfig[ModuleConfig]	org.apache.struts.action.RequestProcessor:moduleConfig[ModuleConfig]	6001	Removed field appConfig	**protected**	Y	1:1	Rename	Was deprecated, replacement described in comments.
org.apache.struts.action.RequestProcessor.processActionForward(HttpServletRequest, HttpServletResponse, ActionForward)[**void**]	org.apache.struts.action.RequestProcessor.processForwardConfig(HttpServletRequest, HttpServletResponse, ActionForward)[**void**]	7002	Method '**protected void** processActionForward(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse, org.apache.struts.action.ActionForward)' has been removed	**protected**	Y	1:1	Rename	Was deprecated, replacement described in comments.
org.apache.struts.action.RequestProcessor.getDebug()[**int**]	none	7002	Method '**public int** getDebug()' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacment partially described in comments, and release notes. Essentially, you're expected to either (a) utilize your own logging now, or (b) utilize the commons logging framework. There's some straightfoward replacments using LOG4J for this code.
org.apache.struts.config.ActionConfig.getApplicationConfig()[ModuleConfig]	org.apache.struts.config.ActionConfig.getModuleConfig()[ModuleConfig]	7002	Method '**public** org.apache.struts.config.ModuleConfig getApplicationConfig()' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement described in comments.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ActionConfig.setApplicationConfig(ModuleConfig)[**void**]	org.apache.struts.config.ActionConfig.setModuleConfig(ModuleConfig)[**void**]	7002	Method '**public void** setApplicationConfig(org.apache.struts.config.ModuleConfig)' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement described in comments.
org.apache.struts.config.ApplicationConfig	org.apache.struts.config.impl.ModuleConfigImpl	8001	Class org.apache.struts.config.ApplicationConfig removed	**public**	Y	1:1	Shorten subtype chain	Deprecated in previous version, now deleted. Comments indicated it should be replaced by the ModuleConfig interface, but since this was a subclass of ModuleConfigImpl, that's a better, more direct replacement.
org.apache.struts.config.ConfigHelper.errorsEmpty()[**boolean**]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.size(Globals.ERROR_KEY) > 0	7002	Method '**public boolean** errorsEmpty()' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelper.getErrorsEmpty()[**boolean**]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.size(Globals.ERROR_KEY) > 0	7002	Method '**public boolean** getErrorsEmpty()' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n
org.apache.struts.config.ConfigHelper.errorSize()[**int**]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.size(Globals.ERROR_KEY)	7002	Method '**public int** errorSize()' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelper.errorSize(String)[**int**]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getKey().equals(String), count matches manually.	7002	Method '**public int** errorSize(java.lang.String)' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. Note that the get(String) method will only return an Iterator, and there's no easy way to get the size of items iterated on, so a manual count implementation will be needed. 1:n
org.apache.struts.config.ConfigHelper.getErrorSize()[**int**]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.size(Globals.ERROR_KEY)	7002	Method '**public int** getErrorSize()' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelper.getErrorSize(String)[int]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getKey().equals(String), count matches manually.	7002	Method 'public int getErrorSize(java.lang.String)' has been removed	public	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. Note that the get(String) method will only return an Iterator, and there's no easy way to get the size of items iterated on, so a manual count implementation will be needed. 1:n
org.apache.struts.config.ConfigHelper.errorOutput()[String]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getValues()[Object[]], manually joining output	7002	Method 'public java.lang.String errorOutput()' has been removed	public	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. Note that the original method had some formatting code present, this would need to be replicated. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelper.errorOutput(String)[String]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getKey().equals(String), if matched org.apache.struts.action.ActionMessage.getValues[Object[]], manually join values.	7002	Method 'public java.lang.String errorOutput(java.lang.String)' has been removed	public	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. Note that the original method had some formatting code present, this would need to be replicated. 1:n
org.apache.struts.config.ConfigHelper.getErrorOutput()[String]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getValues()[Object[]], manually joining output	7002	Method 'public java.lang.String getErrorOutput()' has been removed	public	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. Note that the original method had some formatting code present, this would need to be replicated. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelper.getErrorOutput(String)[String]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getKey().equals(String), if matched org.apache.struts.action.ActionMessage.getValues[Object[]], manually join values.	7002	Method '**public** java.lang.String getErrorOutput(java.lang.String)' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. Note that the original method had some formatting code present, this would need to be replicated. 1:n
org.apache.struts.config.ConfigHelper.errors()[Iterator]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY)[Iterator]	7002	Method '**public** java.util.Iterator errors()' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelper.errors(String)[Iterator]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getKey().equals(String), if matched org.apache.struts.action.ActionMessage.getValues[Object[]], convert into java.util.List, obtain Iterator from List	7002	Method '**public** java.util.Iterator errors(java.lang.String)' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n
org.apache.struts.config.ConfigHelper.getErrors()[Iterator]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY)[Iterator]	7002	Method '**public** java.util.Iterator getErrors()' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelper.getErrors(String)[Iterator]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getKey().equals(String), if matched org.apache.struts.action.ActionMessage.getValues[Object[]], convert into java.util.List, obtain Iterator from List	7002	Method '**public** java.util.Iterator getErrors(java.lang.String)' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n
org.apache.struts.config.ConfigHelper.getLocale()[Locale]	org.apache.struts.util.RequestUtils.getUserLocale(HttpServletRequest, String)[Locale]	7002	Method '**public** java.util.Locale getLocale()' has been removed	**public**	N	1:n	Move; Rename; Add contextual data	Was deprecated, replacement partially described in comments
org.apache.struts.config.ConfigHelper.getActionErrors()[ActionErrors]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY)	7002	Method '**public** org.apache.struts.action.ActionErrors getActionErrors()' has been removed	**public**	N	1:n	Change Type	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelper.getActionFormBeans()[ActionFormBeans]	org.apache.struts.config.ModuleConfig.findFormBeanConfigs()[FormBeanConfig[]]	7002	Method '**public** org.apache.struts.action.ActionFormBeans getActionFormBeans()' has been removed	**public**	N	1:n	Change Type	Not deprecated, no replacement offered. Deprecated comments in the ActionFormBeans and Globals classes indicated that this is replaced by FormBeanConfig collections in the ModuleConfig class. FormBeanConfig is a supertype of ActionConfig.
org.apache.struts.config.ConfigHelper.getActionForwards()[ActionForwards]	org.apache.struts.config.ModuleConfig.findForwardConfigs()[ForwardConfig[]]	7002	Method '**public** org.apache.struts.action.ActionForwards getActionForwards()' has been removed	**public**	N	1:n	Change Type	Not deprecated, no replacement offered. Deprecated comments in the ActionForwards and Globals classes indicated that this is replaced by ForwardConfig collections in the ModuleConfig class. ForwardConfig is a supertype of ActionConfig.
org.apache.struts.config.ConfigHelper.getActionMappings()[ActionMappings]	org.apache.struts.config.ModuleConfig.findActionConfigs()[ActionConfig[]]	7002	Method '**public** org.apache.struts.action.ActionMappings getActionMappings()' has been removed	**public**	N	1:n	Change Type	Not deprecated, no replacement offered. Deprecated comments in the ActionForwards and Globals classes indicated that this is replaced by ForwardConfig collections in the ModuleConfig class. ForwardConfig is a supertype of ActionConfig.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelperInterface.getErrorsEmpty()[**boolean**]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.size(Globals.ERROR_KEY) > 0	7002	Method '**public boolean** getErrorsEmpty()' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n
org.apache.struts.config.ConfigHelperInterface.getErrorSize()[**int**]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.size(Globals.ERROR_KEY)	7002	Method '**public int** getErrorSize()' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelperInterface.getErrorSize(String)[**int**]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getKey().equals(String), count matches manually.	7002	Method '**public int** getErrorSize(java.lang.String)' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. Note that the get(String) method will only return an Iterator, and there's no easy way to get the size of items iterated on, so a manual count implementation will be needed. 1:n
org.apache.struts.config.ConfigHelperInterface.getErrorOutput()[String]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getValues()[Object[]], manually joining output	7002	Method '**public** java.lang.String getErrorOutput()' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. Note that the original method had some formatting code present, this would need to be replicated. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelperInterface.getErrorOutput(String)[String]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getKey().equals(String), if matched org.apache.struts.action.ActionMessage.getValues[Object[]], manually join values.	7002	Method '**public** java.lang.String getErrorOutput(java.lang.String)' has been removed	**public**	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. Note that the original method had some formatting code present, this would need to be replicated. 1:n

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelperInterface.getErrors()[Iterator]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY)[Iterator]	7002	Method 'public java.util.Iterator getErrors()' has been removed	public	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n
org.apache.struts.config.ConfigHelperInterface.getErrors(String)[Iterator]	org.apache.struts.config.ConfigHelper.getActionMessages()[ActionMessages]; org.apache.struts.action.ActionMessages.get(Globals.ERROR_KEY); java.util.Iterator over org.apache.struts.action.ActionMessage.getKey().equals(String), if matched org.apache.struts.action.ActionMessage.getValues[Object[]], convert into java.util.List, obtain Iterator from List	7002	Method 'public java.util.Iterator getErrors(java.lang.String)' has been removed	public	N	1:n	Change Type; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Release notes indicate they're moving from ActionErrors to ActionMessages, so this may be the proper way to get this now. 1:n
org.apache.struts.config.ConfigHelperInterface.getLocale()[Locale]	org.apache.struts.util.RequestUtils.getUserLocale(HttpServletRequest, String)[Locale]	7002	Method 'public java.util.Locale getLocale()' has been removed	public	N	1:n	Move; Rename; Add contextual data	Not deprecated, no replacement offered.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelperInterface.getActionErrors()[ActionErrors]	org.apache.struts.config.ConfigHelperInterface.getActionMessages()[ActionMessages]	7002	Method '**public** org.apache.struts.action.ActionErrors getActionErrors()' has been removed	**public**	N	1:1	Shorten subtype chain	Not deprecated, no replacement offered. Release notes indicated that ActionErrors should be replaced with ActionMessages, its supertype.
org.apache.struts.config.ConfigHelperInterface.getActionFormBeans()[ActionFormBeans]	org.apache.struts.config.ModuleConfig.findFormBeanConfigs()[FormBeanConfig[]]	7002	Method '**public** org.apache.struts.action.ActionFormBeans getActionFormBeans()' has been removed	**public**	N	1:n	Change Type	Not deprecated, no replacement offered. Deprecated comments in the ActionFormBeans and Globals classes indicated that this is replaced by FormBeanConfig collections in the ModuleConfig class. FormBeanConfig is a supertype of ActionConfig.
org.apache.struts.config.ConfigHelperInterface.getActionForwards()[ActionForwards]	org.apache.struts.config.ModuleConfig.findForwardConfigs()[ForwardConfig[]]	7002	Method '**public** org.apache.struts.action.ActionForwards getActionForwards()' has been removed	**public**	N	1:n	Change Type	Not deprecated, no replacement offered. Deprecated comments in the ActionForwards and Globals classes indicated that this is replaced by ForwardConfig collections in the ModuleConfig class. ForwardConfig is a supertype of ActionConfig.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ConfigHelperInterface.getActionMappings()[ActionMappings]	org.apache.struts.config.ModuleConfig.findActionConfigs()[ActionConfig[]]	7002	Method '**public** org.apache.struts.action.ActionMappings getActionMappings()' has been removed	**public**	N	1:n	Change Type	Not deprecated, no replacement offered. Deprecated comments in the ActionMappings and Globals classes indicated that this is replaced by ActionConfig collections in the ModuleConfig class.
org.apache.struts.config.ControllerConfig:debug[unknown]	none	6001	Removed field debug	**protected**	Y	1:n	Extract to External API	Was deprecated, no replacement offered. Release notes from an earlier version indicated that they were eliminating logging functionality in STRUTS, and expect callers to use their own logging functionality, or use other loggers like LOG4J (also from Apache); there's straightforward replacements from that library.
org.apache.struts.config.ControllerConfig.getDebug()[**int**]	none	7002	Method '**public int** getDebug()' has been removed	**public**	N	1:n	Extract to External API	Not deprecated, no replacement offered. Field this method was dependent on was deprecated. Callers are expected to use another library for logging.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.ControllerConfig.setDebug(**int**)[**void**]	none	7002	Method '**public void** setDebug(**int**)' has been removed	**public**	N	1:n	Extract to External API	Not deprecated, no replacement offered. Field this method was dependent on was deprecated. Callers are expected to use another library for logging.
org.apache.struts.config.FormBeanConfig:moduleConfig[ModuleConfig]	org.apache.struts.util.ModuleUtils.getModuleConfig(HttpServletRequest)[ModuleConfig]; javax.util.HttpServletRequest.setAttribute(Globals.MODULE_KEY, ModuleConfig)	6001	Removed field moduleConfig	**protected**	N	1:n	Move; Encapsulate; Add Contextual Data; Expose/Unpack Implementation	Not deprecated in old version, no replacement offered. Experience led me to check out the Globals file and look at the MODULE_KEY constant, which lead me to ModuleUtils. Note that ModuleUtils doesn't offer support for setting the ModuleConfig now, so if you need to do that it's by hand.
org.apache.struts.config.FormBeanConfig.getModuleConfig()[ModuleConfig]	org.apache.struts.util.ModuleUtils.getModuleConfig(HttpServletRequest)[ModuleConfig]	7002	Method '**public** org.apache.struts.config.ModuleConfig getModuleConfig()' has been removed	**public**	N	1:n	Move; Add Contextual Data	Not deprecated in old version, no replacement offered. A similar convenience method to this is offered in org.struts.util.ModuleUtils.
org.apache.struts.config.FormBeanConfig.setModuleConfig(ModuleConfig)[**void**]	javax.util.HttpServletRequest.setAttribute(Globals.MODULE_KEY, ModuleConfig)	7002	Method '**public void** setModuleConfig(org.apache.struts.config.ModuleConfig)' has been removed	**public**	N	1:1	Expose/Unpack Implementation	Not deprecated in old version, no replacement offered.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
none	org.apache.struts.config.ModuleConfig.getActionFormBeanClass()[String]	7012	Method '**public** java.lang.String getActionFormBeanClass()' has been added to an interface	**public**	N	1:n	Add to Interface	
none	org.apache.struts.config.ModuleConfig.getActionForwardClass()[String]	7012	Method '**public** java.lang.String getActionForwardClass()' has been added to an interface	**public**	N	1:n	Add to Interface	
none	org.apache.struts.config.ModuleConfig.setActionFormBeanClass(String)[**void**]	7012	Method '**public void** setActionFormBeanClass(java.lang.String)' has been added to an interface	**public**	N	1:n	Add to Interface	
none	org.apache.struts.config.ModuleConfig.setActionForwardClass(String)[**void**]	7012	Method '**public void** setActionForwardClass(java.lang.String)' has been added to an interface	**public**	N	1:n	Add to Interface	

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.config.impl. ModuleConfigImpl. ModuleConfigImpl(ModuleConfigImpl)	org.apache.struts.config.impl. ModuleConfigImpl. ModuleConfigImpl(); org.apache.struts.config.impl. ModuleConfigImpl.setPrefix(moduleConfigImple.getPrefix); org.apache.struts.config.impl. ModuleConfigImpl.setController(ModuleConfigImpl.getController)	7002	Method '**public** ModuleConfigImpl(org. apache.struts.config.impl. ModuleConfigImpl)' has been removed	**public**	Y	1:n	Remove Parameter; Expose/Un-pack Implementa-tion	Was deprecated, no replacement offered. This constructor shallow copied a ModuleConfigImpl; this replacement is a partial solution given that not all fields have setters. A better replacement would be to subclass, and reimplement the constructor, or subclass and implement Cloneable, then clone() the object.
org.apache.struts.taglib.html. FormTag:name[String]	org.apache.struts.taglib.html. FormTag:beanName[String]	6001	Removed field name	**protected**	N	1:1	Rename	Deprecated in previous version, now deleted. Release notes indicate that values stored in this field should be present in a config file (struts-config.xml); comment with this field indicates it is the replacement.
org.apache.struts.taglib.html. FormTag:scope[String]	org.apache.struts.taglib.html. FormTag:beanScope[String]	6001	Removed field scope	**protected**	N	1:1	Rename	Deprecated in previous version, now deleted. Release notes indicate that values stored in this field should be present in a config file (struts-config.xml); comment with this field indicates it is the replacement.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.html.FormTag:type[String]	org.apache.struts.taglib.html.FormTag:beanType[String]	6001	Removed field type	**protected**	N	1:1	Rename	Deprecated in previous version, now deleted. Release notes indicate that values stored in this field should be present in a config file (struts-config.xml); comment with this field indicates it is the replacement.
org.apache.struts.taglib.html.FormTag.getName()[String]	javax.servlet.jsp.tagext.TagSupport.getValue(String)[Object]	7002	Method '**public** java.lang.String getName()' has been removed	**public**	N	1:n	Extract to External API; Change Type	The functionality for getting the values stored in the configuration file, which are dynamically loaded into this object on creation, is to use these methods in the superclass TagSupport.
org.apache.struts.taglib.html.FormTag.getScope()[String]	javax.servlet.jsp.tagext.TagSupport.getValue(String)[Object]	7002	Method '**public** java.lang.String getScope()' has been removed	**public**	N	1:n	Extract to External API; Change Type	The functionality for getting the values stored in the configuration file, which are dynamically loaded into this object on creation, is to use these methods in the superclass TagSupport.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.html.FormTag.getType()[String]	javax.servlet.jsp.tagext.TagSupport.getValue(String)[Object]	7002	Method '**public** java.lang.String getType()' has been removed	**public**	N	1:n	Extract to External API; Change Type	The functionality for getting the values stored in the configuration file, which are dynamically loaded into this object on creation, is to use these methods in the superclass TagSupport.
org.apache.struts.taglib.html.FormTag.setName(String)[**void**]	javax.servlet.jsp.tagext.TagSupport.setValue(String, Object)[**void**]	7002	Method '**public void** setName(java.lang.String)' has been removed	**public**	N	1:n	Extract to External API; Change Type	

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.html.FormTag.setScope(String)[**void**]	javax.servlet.jsp.tagext.TagSupport.setValue(String, Object)[**void**]	7002	Method '**public void** setScope(java.lang.String)' has been removed	**public**	N	1:n	Extract to External API; Change Type	
org.apache.struts.taglib.html.FormTag.setType(String)[**void**]	javax.servlet.jsp.tagext.TagSupport.setValue(String, Object)[**void**]	7002	Method '**public void** setType(java.lang.String)' has been removed	**public**	N	1:n	Extract to External API; Change Type	
org.apache.struts.taglib.html.FrameTag:marginheight[**int**]	org.apache.struts.taglib.html.FrameTag:marginheight[Integer]	6004	Changed type of field marginheight from **int** to java.lang.Integer	**protected**	N	1:n	Change Type	Wasn't deprecated. Not sure if they expected the compiler's auto-unboxing/boxing to automatically handle this.
org.apache.struts.taglib.html.FrameTag:marginwidth[**int**]	org.apache.struts.taglib.html.FrameTag:marginheight[Integer]	6004	Changed type of field marginwidth from **int** to java.lang.Integer	**protected**	N	1:n	Change Type	Wasn't deprecated. Not sure if they expected the compiler's auto-unboxing/boxing to automatically handle this.
org.apache.struts.taglib.html.FrameTag.getMarginheight()[**int**]	org.apache.struts.taglib.html.FrameTag.getMarginheight()[Integer]	7006	Return type of method '**public int** getMarginheight()' has been changed to java.lang.Integer	**public**	N	1:n	Change Type	Wasn't deprecated. Not sure if they expected the compiler's auto-unboxing/boxing to automatically handle this.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.html.FrameTag.getMarginwidth()[**int**]	org.apache.struts.taglib.html.FrameTag.getMarginwidth()[Integer]	7006	Return type of method '**public int** getMarginwidth()' has been changed to java.lang.Integer	**public**	N	1:n	Change Type	Wasn't deprecated. Not sure if they expected the compiler's auto-unboxing/boxing to automatically handle this.
org.apache.struts.taglib.html.FrameTag.setMarginheight(**int**)[**void**]	org.apache.struts.taglib.html.FrameTag.setMarginheight(Integer)[**void**]	7005	Parameter 1 of '**public void** setMarginheight(**int**)' has changed its type to java.lang.Integer	**public**	N	1:n	Change Type	Wasn't deprecated. Not sure if they expected the compiler's auto-unboxing/boxing to automatically handle this.
org.apache.struts.taglib.html.FrameTag.setMarginwidth(**int**)[**void**]	org.apache.struts.taglib.html.FrameTag.setMarginwidth(Integer)[**void**]	7005	Parameter 1 of '**public void** setMarginwidth(**int**)' has changed its type to java.lang.Integer	**public**	N	1:n	Change Type	Wasn't deprecated. Not sure if they expected the compiler's auto-unboxing/boxing to automatically handle this.
org.apache.struts.taglib.html.HtmlTag.currentLocale()[Locale]	org.apache.struts.taglib.html.HtmlTag.getCurrentLocale()[Locale]	7002	Method '**protected** java.util.Locale currentLocale()' has been removed	**protected**	Y	1:1	Rename	Was deprecated, replacement offered.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.nested.html.NestedErrorsTag	org.apache.struts.taglib.nested.html.NestedErrorsTag	4001	Removed org.apache.struts.taglib.nested.NestedNameSupport from the set of implemented interfaces	**public**	N	1:1	Supertype Change; Change Type	Wasn't deprecated. This eliminated the lowest implement sub-interface from hierarchy of sub-interfaces, i.e., NestedNameSupport is a subclass/subinterface of NestedPropertySupport, which is now implemented in this class instead. This is a weird implementation though: this class never implemented the methods in the interface, its superclass did (although the super class DOESN'T extend the interface), but with the removal of the interface those methods are still present in the superclass. If this interface had been left in, nothing would have changed.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.nested.html.NestedMessagesTag	org.apache.struts.taglib.nested.html.NestedMessagesTag	4001	Removed org.apache.struts.taglib.nested.NestedNameSupport from the set of implemented interfaces	**public**	N	1:1	Supertype Change; Change Type	Wasn't deprecated. This eliminated the lowest implement sub-interface from hierarchy of sub-interfaces, i.e., NestedNameSupport is a subclass/subinterface of NestedPropertySupport, which is now implemented in this class instead. This is a weird implementation though: this class never implemented the methods in the interface, its superclass did (although the super class DOESN'T extend the interface), but with the removal of the interface those methods are still present in the superclass. If this interface had been left in, nothing would have changed.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.nested.logic.NestedMessagesNotPresentTag	org.apache.struts.taglib.nested.logic.NestedMessagesNotPresentTag	4001	Removed org.apache.struts.taglib.nested.NestedNameSupport from the set of implemented interfaces	**public**	N	1:1	Supertype Change; Change Type	Wasn't deprecated. This eliminated the lowest implement sub-interface from hierarchy of sub-interfaces, i.e., NestedNameSupport is a subclass/subinterface of NestedPropertySupport, which is now implemented in this class instead. This is a weird implementation though: this class never implemented the methods in the interface, one of its superclass did (although that super class DIDN'T extend the interface), but with the removal of the interface those methods are still present in that superclass. If this interface had been left in, nothing would have changed.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.nested.logic.NestedMessagesPresentTag	org.apache.struts.taglib.nested.logic.NestedMessagesPresentTag	4001	Removed org.apache.struts.taglib.nested.NestedNameSupport from the set of implemented interfaces	**public**	N	1:1	Supertype Change; Change Type	Wasn't deprecated. This eliminated the lowest implement sub-interface from hierarchy of sub-interfaces, i.e., NestedNameSupport is a subclass/subinterface of NestedPropertySupport, which is now implemented in this class instead. This is a weird implementation though: this class never implemented the methods in the interface, one of its superclass did (although that super class DIDN'T extend the interface), but with the removal of the interface those methods are still present in that superclass. If this interface had been left in, nothing would have changed.
org.apache.struts.taglib.template.GetTag	org.apache.struts.taglib.tiles.GetTag	8001	Class org.apache.struts.taglib.template.GetTag removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered.
org.apache.struts.taglib.template.InsertTag	org.apache.struts.taglib.tiles.InsertTag	8001	Class org.apache.struts.taglib.template.InsertTag removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.template.PutTag	org.apache.struts.taglib.tiles.PutTag	8001	Class org.apache.struts.taglib.template.PutTag removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered.
org.apache.struts.taglib.template.util.Content	org.apache.struts.taglib.tiles.PutTag	8001	Class org.apache.struts.taglib.template.util.Content removed	**public**	Y	1:n	Change Type	Was deprecated, no replacement offered. Comments simply say "Use Tiles instead", but no direction on what a substitute would be—in fact, there appears to be no substitute.
org.apache.struts.taglib.template.util.ContentMap	java.util.HashMap	8001	Class org.apache.struts.taglib.template.util.ContentMap removed	**public**	Y	1:n	Expose/Unpack Implementation	Was deprecated, no replacement offered. Comments simply say "Use Tiles instead", but no direction on what a substitute would be—in fact, there appears to be no substitute.
org.apache.struts.taglib.template.util.ContentMapStack	java.util.Stack	8001	Class org.apache.struts.taglib.template.util.ContentMapStack removed	**public**	Y	1:n	Expose/Unpack Implementation	Was deprecated, no replacement offered. Comments simply say "Use Tiles instead", but no direction on what a substitute would be—in fact, there appears to be no substitute.
org.apache.struts.taglib.tiles.InsertTag.setInstance(String)[**void**]	org.apache.struts.taglib.tiles.InsertTag.setDefinition(String)[**void**]	7002	Method '**public void** setInstance(java.lang.String)' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement offered.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.tiles.ext.TextTag	org.apache.struts.taglib.tiles.html.TextTag	8001	Class org.apache.struts.taglib.tiles.ext.TextTag removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered. This is a 1:n replacement; this was a convenience class meant to predefine a bunch of input tags on text fields, replacing it would require doing this configuration in the calling code.
org.apache.struts.tiles.ActionComponentServlet	org.apache.struts.action.ActionServlet; org.apache.struts.tiles.TilesPlugin ; org.apache.struts.tiles.TilesRequestProcessor	8001	Class org.apache.struts.tiles.ActionComponentServlet removed	public	Y	1:n	Change Type	Was deprecated, replacement offered. This is a 1:n replacment; the Tiles package was previously separate from STRUTS, it's now integrated, but to use it some config options need to be set, specifically the org.apache.struts.tiles.TilesPlugin and org.apache.struts.tiles.TilesRequestProcessor need to be specified in the configuration to override the normal Plugins/RequestProcessors.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles. ComponentContext. ComponentContext(ComponentDefinition)	org.apache.struts.tiles. ComponentDefinition. getAttributes(); org.apache.struts.tiles. ComponentContext. ComponentContext(Map)	7002	Method '**public** ComponentContext(org.apache. struts.tiles.ComponentDefinition)' has been removed	**public**	Y	1:n	Change Type; Add Contextual Data	Was deprecated, replacement offered. This is a 1:n replacement; basically implementing what the constructor did, which was a wrapper/delegate to another constructor.
none	org.apache.struts.tiles.Controller. execute(ComponentContext, HttpServletRequest, HttpServletResponse, ServletContext)[**void**]	7012	Method '**public void** execute(org.apache.struts.tiles. ComponentContext, javax.servlet. http.HttpServletRequest, javax. servlet.http.HttpServletResponse, javax.servlet.ServletContext)' has been added to an interface	**public**	N	1:n	Add to Interface	This is meant to replace org.apache .struts.tiles.Controller.perform(ComponentContext, HttpServletRequest , HttpServletResponse, ServletContext)[**void**]
org.apache.struts.tiles. DefinitionAttribute	org.apache.struts.tiles. DefinitionAttribute	5001	Removed org.apache.struts. tiles.UntyppedAttribute from the list of superclasses	**public**	N	1:1	Supertype Change	Wasn't deprecated, no replacement offered. This is a side effect of the supertype having a typo in the class name in the old version, and having it fixed in the new.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles.DefinitionNameAttribute	org.apache.struts.tiles.DefinitionNameAttribute	5001	Removed org.apache.struts.tiles.UntyppedAttribute from the list of superclasses	**public**	N	1:1	Supertype Change	Wasn't deprecated, no replacement offered. This is a side effect of the supertype having a typo in the class name in the old version, and having it fixed in the new.
org.apache.struts.tiles.DefinitionsFactoryConfig: debugLevel[unknown]	none	6001	Removed field debugLevel	**protected**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.
org.apache.struts.tiles.DefinitionsFactoryConfig: parserDebugLevel[unknown]	none	6001	Removed field parserDebugLevel	**protected**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles.DefinitionsFactoryConfig.getDebugLevel()[**int**]	none	7002	Method '**public int** getDebugLevel()' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.
org.apache.struts.tiles.DefinitionsFactoryConfig.getParserDebugLevel()[**int**]	none	7002	Method '**public int** getParserDebugLevel()' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.
org.apache.struts.tiles.DefinitionsFactoryConfig.setDebugLevel(**int**)[**void**]	none	7002	Method '**public void** setDebugLevel(**int**)' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles.DefinitionsFactoryConfig.setParserDebugLevel(**int**)[**void**]	none	7002	Method '**public void** setParserDebugLevel(**int**)' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles.DefinitionsUtil.setDefinitionsFactory(ComponentDefinitionsFactory, ServletContext)[**void**]	org.apache.struts.tiles.DefinitionsFactory.getConfig(); org.apache.struts.tiles.TilesUtil.createDefinitionsFactory(ServletContext, DefinitionsFactoryConfig)[DefinitionsFactory]	7002	Method '**protected void** setDefinitionsFactory(org.apache.struts.tiles.ComponentDefinitionsFactory, javax.servlet.ServletContext)' has been removed	**protected**	Y	1:n	Change Type; Add Contextual Data; Remap	Was deprecated, wrong replacement offered. This is a 1:n replacement; the replacement offered was deprecated in the old version, and removed in the new as well. Some work is needed to convert the ComponentDefinitionsFactory object to a DefinitionsFactory object before this method can be called. What's interesting in this case is that the code comment which directs you to the replacement is wrong, but if you look at the specified replacement in the older release, its code comment will lead you to the correct solution. So an erroneous comment was better than nothing in this case.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles.DefinitionsUtil.getDefinition(String, PageContext)[ComponentDefinition]	javax.servlet.jsp.PageContext.getRequest(); javax.servlet.jsp.PageContext.getServletContext(); org.apache.struts.tiles.TilesUtil.getDefinition(String, ServletRequest, ServletContext)[ComponentDefinition]	7002	Method '**public** org.apache.struts.tiles.ComponentDefinition getDefinition(java.lang.String, javax.servlet.jsp.PageContext)' has been removed	**public**	Y	1:n	Change Type; Add Contextual Data; Remap	Was deprecated, replacement offered. This is a 1:n replacement; some work is needed to modify the parameters so they match the new target method.
org.apache.struts.tiles.DefinitionsUtil.createDefinitionFactoryInstance(String)[DefinitionsFactory]	org.apache.struts.tiles.DefinitionsFactory.getConfig(); org.apache.struts.tiles.TilesUtil.createDefinitionsFactory(ServletContext, DefinitionsFactoryConfig)[DefinitionsFactory]	7002	Method '**public** org.apache.struts.tiles.DefinitionsFactory createDefinitionFactoryInstance(java.lang.String)' has been removed	**public**	Y	1:n	Change Type; Add Contextual Data; Remap	Was deprecated, replacement offered. This is a 1:n replacement; developer comments notes that there isn't a direct replacement, but this is what it should be replaced with, so some work will be needed to make it fit.
org.apache.struts.tiles.DefinitionsUtil.initUserDebugLevel(ServletConfig)[**void**]	none	7002	Method '**public void** initUserDebugLevel(javax.servlet.ServletConfig)' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles. DefinitionsUtil. makeDefinitionsFactoryAccessible (DefinitionsFactory, ServletContext)[**void**]	org.apache.struts.tiles. DefinitionsFactory.getConfig(); org.apache.struts.tiles.TilesUtil. createDefinitionsFactory(ServletContext, DefinitionsFactoryConfig)[DefinitionsFactory]	7002	Method '**public void** makeDefinitionsFactoryAccessible (org.apache.struts.tiles. DefinitionsFactory, javax.servlet. ServletContext)' has been removed	**public**	Y	1:n	Change Type; Add Contextual Data; Remap	Was deprecated, replacement offered. This is a 1:n replacement; some work is needed to modify the parameters so they match the new target method.
org.apache.struts.tiles. DefinitionsUtil. setUserDebugLevel(**int**)[**void**]	none	7002	Method '**public void** setUserDebugLevel(**int**)' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.
org.apache.struts.tiles. DirectStringAttribute	org.apache.struts.tiles. DirectStringAttribute	5001	Removed org.apache.struts. tiles.UntyppedAttribute from the list of superclasses	**public**	N	1:1	Supertype Change	Wasn't deprecated, no replacement offered. This is a side effect of the supertype having a typo in the class name in the old version, and having it fixed in the new.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles.PathAttribute	org.apache.struts.tiles.PathAttribute	5001	Removed org.apache.struts.tiles.UntyppedAttribute from the list of superclasses	**public**	N	1:1	Supertype Change	Wasn't deprecated, no replacement offered. This is a side effect of the supertype having a typo in the class name in the old version, and having it fixed in the new.
org.apache.struts.tiles.TilesServlet	org.apache.struts.action.ActionServlet	8001	Class org.apache.struts.tiles.TilesServlet removed	**public**	Y	1:1	Change Type	Was deprecated, replacement offered. This is a 1:n replacment; the Tiles package was previously separate from STRUTS, it's now integrated, but to use it some config options need to be set, specifically the org.apache.struts.tiles.TilesPlugin and org.apache.struts.tiles.TilesRequestProcessor need to be specified in the configuration to override the normal Plugins/RequestProcessors.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles.TilesUtil.applicationClass(String)[Class]	org.apache.struts.util.RequestUtils.applicationClass(String)[Class]	7002	Method '**public** java.lang.Class applicationClass(java.lang.String)' has been removed	**public**	Y	1:1	Move	Was deprecated, replacement offered. In the current code, existing functionality was delegated to another deprecated method (see below), which in turn was delegated to this replacement functionality. Interesting to see if this trips up some tools.
org.apache.struts.tiles.TilesUtilImpl.applicationClass(String)[Class]	org.apache.struts.util.RequestUtils.applicationClass(String)[Class]	7002	Method '**public** java.lang.Class applicationClass(java.lang.String)' has been removed	**public**	Y	1:1	Move	Was deprecated, replacement offered. In the current code, existing functionality was delegated before being removed.
org.apache.struts.tiles.TilesUtilStrutsImpl.applicationClass(String)[Class]	org.apache.struts.util.RequestUtils.applicationClass(String)[Class]	7002	Method '**public** java.lang.Class applicationClass(java.lang.String)' has been removed	**public**	Y	1:1	Move	Was deprecated, replacement offered. In the current code, existing functionality was delegated before being removed.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles. UntyppedAttribute	org.apache.struts.tiles. UntypedAttribute	8001	Class org.apache.struts.tiles. UntyppedAttribute removed	**public**	N	1:1	Rename	Wasn't deprecated, no replacement offered. The original class had a typo in the name—one too many p's. Interesting that they just made this change, without any warning; begs the question of how such changes should be handled, and this would be the perfect case of process vs. expediency.
org.apache.struts.tiles.actions. NoOpAction	org.apache.struts.actions. ForwardAction	8001	Class org.apache.struts.tiles. actions.NoOpAction removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered. This is a 1:n replacement; the comments described some additional configuration needed to make ActionForward behave in the same way.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.tiles.actions.TilesAction.perform(ComponentContext, ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)[ActionForward]	org.apache.struts.tiles.actions.TilesAction.execute(ComponentContext, ActionMapping, ActionForm, HttpServletRequest, HttpServletResponse)[ActionForward]	7002	Method '**public** org.apache.struts.action.ActionForward perform(org.apache.struts.tiles.ComponentContext, org.apache.struts.action.ActionMapping, org.apache.struts.action.ActionForm, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement offered.
org.apache.struts.tiles.xmlDefinition.XmlParser.setDetailLevel(**int**)[**void**]	none	7002	Method '**public void** setDetailLevel(**int**)' has been removed	**public**	Y	1:n	Extract to External API	Was deprecated, replacement partially described. This method was damaged in the old version to do nothing; it's meant to set a logging level, which is now expected to be set in an external logging library.
org.apache.struts.upload.MultipartElement.MultipartElement(String, String, String, **byte**[])	org.apache.struts.upload.MultipartElement.MultipartElement(String, String, String, File)	7002	Method '**public** MultipartElement(java.lang.String, java.lang.String, java.lang.String, **byte**[])' has been removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered. This is a 1:n replacement as the **byte**[] data would need to be written to a temporary file, and then passed as a parameter to the new constructor.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.upload.MultipartElement.getData()[**byte**[]]	org.apache.struts.upload.MultipartElement.getFile()[File]	7002	Method '**public byte**[] getData()' has been removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered. This is a 1:n replacement as the File object needs to be written to a **byte**[] array.
org.apache.struts.upload.MultipartElement.setData(**byte**[])[**void**]	org.apache.struts.upload.MultipartElement.setFile(File)[**void**]	7002	Method '**public void** setData(**byte**[])' has been removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered. This is a 1:n replacement as the **byte**[] data would need to be written to a temporary file, and then passed as a parameter to the new constructor.
org.apache.struts.util.AppException	org.apache.struts.util.ModuleException	8001	Class org.apache.struts.util.AppException removed	**public**	Y	1:1	Shorten Subtype Chain	Was deprecated, replacement offered.
org.apache.struts.util.ErrorMessages	org.apache.struts.action.ActionErrors	8001	Class org.apache.struts.util.ErrorMessages removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered.
org.apache.struts.util.GenericDataSource	none	8001	Class org.apache.struts.util.GenericDataSource removed	**public**	Y	1:n	Change Type; Extract to External API	Was deprecated, replacement partially described. Caller is expected to get accesss to a data source via the J2EE container now, not through STRUTS.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.IteratorAdapter	org.apache.commons.collections.IteratorUtils.asIterator(Enumeration)[Iterator]	8001	Class org.apache.struts.util.IteratorAdapter removed	**public**	Y	1:1	Change Type; Drop Façade	Was deprecated, replacement offered. This is interesting: the current code is an Iterator wrapper over Enumeration, they moved this into an external library. What's interesting is a type is going to be replaced by a **static** method invocation. Would be interesting to see what fails on this.
org.apache.struts.util.RequestUtils.getApplicationPrefixes(ServletContext)[String[]]	org.apache.struts.util.RequestUtils.getModulePrefixes(ServletContext)[String[]]	7002	Method '**public** java.lang.String [] getApplicationPrefixes(javax.servlet.ServletContext)' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement offered. In the current code, existing functionality was delegated before being removed.
org.apache.struts.util.RequestUtils.selectApplication(String, HttpServletRequest, ServletContext)[**void**]	org.apache.struts.util.RequestUtils.selectModule(String, HttpServletRequest, ServletContext)[**void**]	7002	Method '**public void** selectApplication(java.lang.String, javax.servlet.http.HttpServletRequest, javax.servlet.ServletContext)' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement offered. In the current code, existing functionality was delegated before being removed.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.RequestUtils.selectApplication(HttpServletRequest, ServletContext)[**void**]	org.apache.struts.util.RequestUtils.selectModule(HttpServletRequest, ServletContext)[**void**]	7002	Method '**public void** selectApplication(javax.servlet.http.HttpServletRequest, javax.servlet.ServletContext)' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement offered. In the current code, existing functionality was delegated before being removed.
org.apache.struts.util.StrutsValidator	org.apache.struts.validator.FieldChecks	8001	Class org.apache.struts.util.StrutsValidator removed	**public**	Y	1:1	Remap	Was deprecated, replacement offered. In the current code, existing functionality was delegated before being removed.
org.apache.struts.util.StrutsValidatorUtil	org.apache.struts.validator.Resources	8001	Class org.apache.struts.util.StrutsValidatorUtil removed	**public**	Y	1:1	Remap	Was deprecated, replacement offered. In the current code, existing functionality was delegated before being removed.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.util.TokenProcessor.toHex(**byte**[])[String]	org.apache.struts.util.TokenProcessor.toHex(**byte**[])[String]	7009	Accessibility of method '**public** java.lang.String toHex(**byte**[])' has been decreased from **public** to **private**	**public**	N	1:1	Hide	Wasn't deprecated, no replacement offered. Interesting that there was a warning in the documentation about this method not being part of the public API and only for backwards compatibility, yet it wasn't deprecated? Replacement functionality should exist in the JAVA STANDARD LIBRARY I think—just encoding a **byte** array as a hexadecimal String.
org.apache.struts.validator.DynaValidatorForm.log(String)[**void**]	none	7002	Method '**protected void** log(java.lang.String)' has been removed	**protected**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.DynaValidatorForm.log(String, Throwable)[**void**]	none	7002	Method '**protected void** log(java.lang.String, java.lang.Throwable)' has been removed	**protected**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateDoubleRange(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateDoubleRange(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	7005	Parameter 4 of '**public boolean** validateDoubleRange(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateEmail(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateEmail(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	7005	Parameter 4 of '**public boolean** validateEmail(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateFloatRange(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateFloatRange(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	7005	Parameter 4 of '**public boolean** validateFloatRange(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateIntRange(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateIntRange(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	7005	Parameter 4 of '**public boolean** validateIntRange(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateMask(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateMask(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	7005	Parameter 4 of '**public** **boolean** validateMask(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateMaxLength(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateMaxLength(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	7005	Parameter 4 of '**public boolean** validateMaxLength(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateMinLength(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateMinLength(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	7005	Parameter 4 of '**public boolean** validateMinLength(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateRange(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateIntRange(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	7002	Method '**public boolean** validateRange(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has been removed	**public**	Y	1:1	Rename	Was deprecated, replacement offered.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateRequired(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateRequired(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	7005	Parameter 4 of '**public boolean** validateRequired(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateRequiredIf(Object, ValidatorAction, Field, ActionErrors, Validator, HttpServletRequest)[**boolean**]	org.apache.struts.validator.FieldChecks.validateRequiredIf(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7005	Parameter 4 of '**public boolean** validateRequiredIf(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, org.apache.commons.validator.Validator, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateByte(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Byte]	org.apache.struts.validator.FieldChecks.validateByte(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Byte]	7005	Parameter 4 of 'public java.lang.Byte validateByte(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	public	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateByte(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Byte]	org.apache.struts.validator.FieldChecks.validateByte(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Object]	7006	Return type of method 'public java.lang.Byte validateByte(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has been changed to java.lang.Object	public	N	1:1	Change Type	Wasn't deprecated. Return type changed to Object; code indicates that the change was done so that Boolean.FALSE could be returned as a result instead of null if it wasn't valid.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateDouble(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Double]	org.apache.struts.validator.FieldChecks.validateDouble(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Double]	7005	Parameter 4 of 'public java.lang.Double validateDouble(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	public	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateDouble(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Double]	org.apache.struts.validator.FieldChecks.validateDouble(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Object]	7006	Return type of method 'public java.lang.Double validateDouble(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has been changed to java.lang.Object	public	N	1:1	Change Type	Wasn't deprecated. Return type changed to Object; code indicates that the change was done so that Boolean.FALSE could be returned as a result instead of null if it wasn't valid.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateFloat(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Float]	org.apache.struts.validator.FieldChecks.validateFloat(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Float]	7005	Parameter 4 of '**public** java.lang.Float validateFloat(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateFloat(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Float]	org.apache.struts.validator.FieldChecks.validateFloat(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Object]	7006	Return type of method '**public** java.lang.Float validateFloat(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has been changed to java.lang.Object	**public**	N	1:1	Change Type	Wasn't deprecated. Return type changed to Object; code indicates that the change was done so that Boolean.FALSE could be returned as a result instead of **null** if it wasn't valid.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateInteger(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Integer]	org.apache.struts.validator.FieldChecks.validateInteger(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Integer]	7005	Parameter 4 of 'public java.lang.Integer validateInteger(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	public	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateInteger(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Integer]	org.apache.struts.validator.FieldChecks.validateInteger(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Object]	7006	Return type of method 'public java.lang.Integer validateInteger(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has been changed to java.lang.Object	public	N	1:1	Change Type	Wasn't deprecated. Return type changed to Object; code indicates that the change was done so that Boolean.FALSE could be returned as a result instead of null if it wasn't valid.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateCreditCard(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Long]	org.apache.struts.validator.FieldChecks.validateCreditCard(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Long]	7005	Parameter 4 of 'public java.lang.Long validateCreditCard(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	public	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateCreditCard(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Long]	org.apache.struts.validator.FieldChecks.validateCreditCard(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Object]	7006	Return type of method 'public java.lang.Long validateCreditCard(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has been changed to java.lang.Object	public	N	1:1	Change Type	Wasn't deprecated. Return type changed to Object; code indicates that the change was done so that Boolean.FALSE could be returned as a result instead of null if it wasn't valid.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateLong(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Long]	org.apache.struts.validator.FieldChecks.validateLong(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Long]	7005	Parameter 4 of 'public java.lang.Long validateLong(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	public	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of \code{ActionMessages.
org.apache.struts.validator.FieldChecks.validateLong(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Long]	org.apache.struts.validator.FieldChecks.validateLong(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Object]	7006	Return type of method 'public java.lang.Long validateLong(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has been changed to java.lang.Object	public	N	1:1	Change Type	Wasn't deprecated. Return type changed to Object; code indicates that the change was done so that Boolean.FALSE could be returned as a result instead of null if it wasn't valid.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateShort(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Short]	org.apache.struts.validator.FieldChecks.validateShort(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Short]	7005	Parameter 4 of 'public java.lang.Short validateShort(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	public	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateShort(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Short]	org.apache.struts.validator.FieldChecks.validateShort(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Object]	7006	Return type of method 'public java.lang.Short validateShort(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has been changed to java.lang.Object	public	N	1:1	Change Type	Wasn't deprecated. Return type changed to Object; code indicates that the change was done so that Boolean.FALSE could be returned as a result instead of null if it wasn't valid.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateDate(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Date]	org.apache.struts.validator.FieldChecks.validateDate(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Date]	7005	Parameter 4 of 'public java.util.Date validateDate(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has changed its type to org.apache.struts.action.ActionMessages	public	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator.FieldChecks.validateDate(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Date]	org.apache.struts.validator.FieldChecks.validateDate(Object, ValidatorAction, Field, ActionErrors, HttpServletRequest)[Object]	7006	Return type of method 'public java.util.Date validateDate(java.lang.Object, org.apache.commons.validator.ValidatorAction, org.apache.commons.validator.Field, org.apache.struts.action.ActionErrors, javax.servlet.http.HttpServletRequest)' has been changed to java.lang.Object	public	N	1:1	Change Type	Wasn't deprecated. Return type changed to Object; code indicates that the change was done so that Boolean.FALSE could be returned as a result instead of null if it wasn't valid.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator. Resources.initValidator(String, Object, ServletContext, HttpServletRequest, ActionErrors, **int**)[Validator]	org.apache.struts.validator. Resources.initValidator(String, Object, ServletContext, HttpServletRequest, ActionMessages, **int**)[Validator]	7005	Parameter 5 of '**public** org.apache.commons. validator.Validator initValidator(java.lang.String, java.lang.Object, javax.servlet.ServletContext, javax.servlet.http. HttpServletRequest, org.apache. struts.action.ActionErrors, **int**)' has changed its type to org.apache.struts.action. ActionMessages	**public**	N	1:1	Change Type	Is this a BI? A parameter was replaced by it's supertype—ActionErrors is a subtype of ActionMessages.
org.apache.struts.validator. Resources. getValidatorResources(ServletContext)[ValidatorResources]	org.apache.struts.validator. Resources. getValidatorResources(ServletContext, HttpServletRequest)[ValidatorResources]	7002	Method '**public** org.apache.commons. validator.ValidatorResources getValidatorResources(javax. servlet.ServletContext)' has been removed	**public**	Y	1:n	Add contextual data	Was deprecated, replacement partially described; replacement is described, but there's a typo in the ordering of the parameters. Would be interesting to see if some approaches fail on this.
org.apache.struts.validator. Resources. getMessageResources(ServletContext)[MessageResources]	org.apache.struts.validator. Resources. getMessageResources(HttpServletContext)[MessageResources]	7002	Method '**public** org.apache. struts.util.MessageResources getMessageResources(javax. servlet.ServletContext)' has been removed	**public**	Y	1:n	Change Type	Was deprecated, replacement offered.

Continued on next page

Table E.5 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.ValidatorForm.log(String)[**void**]	none	7002	Method '**protected void** log(java.lang.String)' has been removed	**protected**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.
org.apache.struts.validator.ValidatorForm.log(String, Throwable)[**void**]	none	7002	Method '**protected void** log(java.lang.String, java.lang.Throwable)' has been removed	**protected**	Y	1:n	Extract to External API	Was deprecated, replacement partially described; advised to use another logging framework/API as internal logging is being removed from STRUTS. Apache LOG4J has a direct replacement here.

Table E.6: APACHE STRUTS 1.2.4 to 1.2.6 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.html.BaseFieldTag:name[String]	org.apache.struts.taglib.html.BaseInputTag:name[String]	6001	Removed field name	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.ButtonTag:property[String]	org.apache.struts.taglib.html.SubmitTag:property[String]	6001	Removed field property	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.ButtonTag:text[String]	org.apache.struts.taglib.html.SubmitTag:text[String]	6001	Removed field text	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.ButtonTag:value[String]	org.apache.struts.taglib.html.SubmitTag:value[String]	6001	Removed field value	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.CancelTag:messages[MessageResources]	org.apache.struts.taglib.html.SubmitTag:messages[MessageResources]	6001	Removed field messages	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.CancelTag:property[String]	org.apache.struts.taglib.html.SubmitTag:property[String]	6001	Removed field property	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.CancelTag:text[String]	org.apache.struts.taglib.html.SubmitTag:text[String]	6001	Removed field text	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.CancelTag:value[String]	org.apache.struts.taglib.html.SubmitTag:value[String]	6001	Removed field value	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.ImageTag:property[String]	org.apache.struts.taglib.html.SubmitTag:property[String]	6001	Removed field property	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.ResetTag:messages[MessageResources]	org.apache.struts.taglib.html.SubmitTag:messages[MessageResources]	6001	Removed field messages	**protected**	N	1:1	Pull up	Moved into superclass.

Continued on next page

Table E.6 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.html.ResetTag:property[String]	org.apache.struts.taglib.html.SubmitTag:property[String]	6001	Removed field property	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.ResetTag:text[String]	org.apache.struts.taglib.html.SubmitTag:text[String]	6001	Removed field text	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.ResetTag:value[String]	org.apache.struts.taglib.html.SubmitTag:value[String]	6001	Removed field value	**protected**	N	1:1	Pull up	Moved into superclass.
org.apache.struts.taglib.html.ResetTag.**label**()[String]	org.apache.struts.taglib.html.SubmitTag:value[String] OR org.apache.struts.taglib.html.SubmitTag:text[String]	7002	Method '**protected** java.lang.String **label**()' has been removed	**protected**	N	1:n	Expose/Unpack Implementation; Remap	Deleted, wasn't deprecated. The only method calling this method was massively rewritten, so it wasn't needed. The replacement should be SubmitTag:value, but the old code did an additional check in case it was empty, so both would be needed.

Continued on next page

Table E.6 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.taglib.html.ResetTag.renderResetElement(String)[String]	org.apache.struts.taglib.html.SubmitTag.doEndTag()[**int**]	7002	Method '**protected** java.lang.String renderResetElement(java.lang.String)' has been removed	**protected**	N	n:m	Remap	Deleted, wasn't deprecated. The old method had a number of hardcoded Strings to generate a button tag; the new library version seems to have abstracted out much of the parts so that its reusable across multiple classes. This is an n:m replacement, as this method is one of several that's replaced by this method, which also in turn utilizes several new methods.
org.apache.struts.taglib.html.TextareaTag:name[String]	org.apache.struts.taglib.html.BaseInputTag:name[String]	6001	Removed field name	**protected**	N	1:1	Pull up	Moved into superclass.

Table E.7: APACHE STRUTS 1.2.6 to 1.2.7 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateByte(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Object]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateByte(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[Object]	7004	In method '**public** java.lang.Object validateByte(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.
org.apache.struts.validator.FieldChecks.validateCreditCard(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Object]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateCreditCard(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[Object]	7004	In method '**public** java.lang.Object validateCreditCard(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.

Continued on next page

Table E.7 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator. FieldChecks.validateDate(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Object]	org.apache.commons.validator. ValidatorResources. ValidatorResources(String config_xml_url); org.apache.commons.validator. Validator.Validator(ValidatorResources); org.apache.struts.validator. FieldChecks.validateDate(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[Object]	7004	In method '**public** java.lang. Object validateDate(java.lang. Object, .validator.ValidatorAction, .validator.Field, .action. ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.
org.apache.struts.validator. FieldChecks.validateDouble(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Object]	org.apache.commons.validator. ValidatorResources. ValidatorResources(String config_xml_url); org.apache.commons.validator. Validator.Validator(ValidatorResources); org.apache.struts.validator. FieldChecks.validateDouble(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[Object]	7004	In method '**public** java.lang. Object validateDouble(java.lang. Object, .validator.ValidatorAction, .validator.Field, .action. ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.

Continued on next page

Table E.7 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateDoubleRange(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateDoubleRange(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7004	In method '**public boolean** validateDoubleRange(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.
org.apache.struts.validator.FieldChecks.validateEmail(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateEmail(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7004	In method '**public boolean** validateEmail(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.

Continued on next page

Table E.7 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator. FieldChecks.validateFloat(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Object]	org.apache.commons.validator. ValidatorResources. ValidatorResources(String config_xml_url); org.apache.commons.validator. Validator.Validator(ValidatorResources); org.apache.struts.validator. FieldChecks.validateFloat(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[Object]	7004	In method '**public** java.lang. Object validateFloat(java.lang. Object, .validator.ValidatorAction, .validator.Field, .action. ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.
org.apache.struts.validator. FieldChecks.validateFloatRange(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	org.apache.commons.validator. ValidatorResources. ValidatorResources(String config_xml_url); org.apache.commons.validator. Validator.Validator(ValidatorResources); org.apache.struts.validator. FieldChecks.validateFloatRange(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7004	In method '**public boolean** validateFloatRange(java.lang. Object, .validator.ValidatorAction, .validator.Field, .action. ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.

Continued on next page

Table E.7 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateInteger(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Object]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateInteger(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[Object]	7004	In method '**public** java.lang.Object validateInteger(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.
org.apache.struts.validator.FieldChecks.validateIntRange(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateIntRange(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7004	In method '**public boolean** validateIntRange(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.

Continued on next page

Table E.7 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateLong(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Object]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateLong(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[Object]	7004	In method '**public** java.lang.Object validateLong(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.
org.apache.struts.validator.FieldChecks.validateMask(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateMask(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7004	In method '**public boolean** validateMask(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.

Continued on next page

Table E.7 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateMaxLength(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateMaxLength(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7004	In method '**public boolean** validateMaxLength(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.
org.apache.struts.validator.FieldChecks.validateMinLength(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateMinLength(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7004	In method '**public boolean** validateMinLength(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.

Continued on next page

Table E.7 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator.FieldChecks.validateRequired(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateRequired(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7004	In method '**public boolean** validateRequired(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.
org.apache.struts.validator.FieldChecks.validateShort(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[Object]	org.apache.commons.validator.ValidatorResources.ValidatorResources(String config_xml_url); org.apache.commons.validator.Validator.Validator(ValidatorResources); org.apache.struts.validator.FieldChecks.validateShort(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[Object]	7004	In method '**public** java.lang.Object validateShort(java.lang.Object, .validator.ValidatorAction, .validator.Field, .action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.

Continued on next page

Table E.7 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.struts.validator. FieldChecks.validateUrl(Object, ValidatorAction, Field, ActionMessages, HttpServletRequest)[**boolean**]	org.apache.commons.validator. ValidatorResources. ValidatorResources(String config_xml_url); org.apache.commons.validator. Validator.Validator(ValidatorResources); org.apache.struts.validator. FieldChecks.validateUrl(Object, ValidatorAction, Field, ActionMessages, Validator, HttpServletRequest)[**boolean**]	7004	In method '**public boolean** validateUrl(java. lang.Object, .validator. ValidatorAction, .validator.Field, . action.ActionMessages, HttpServletRequest)' the number of arguments has changed	**public**	N	1:n	Add contextual data	New parameter added; requires the addition of a new library (apache.commons.validator), and two lines of setup code to create and configure the parameter that needs to be passed.

Table E.8: JDOM b6 to b7 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.CDATA.setText(String)[**void**]	org.jdom.CDATA.CDATA(String)	7002	Method '**public void** setText(java.lang.String)' has been removed	**public**	Y	1:n	Immutable; Remap	The intention of the class was that the data it contains be immutable, so after class creation it can't be altered. Instead, a new CDATA object would need to be created.
org.jdom.Document:rootElement[Element]	org.jdom.Document:content[List]	6001	Removed field rootElement	**protected**	N	1:n	Remap	Instead of keeping a single root element, a list of content objects is kept, and the first Element object in the list is considered to be the root element.
org.jdom.Document.addContent(Element)[Document]	org.jdom.Document.setRootElement(Element):Document	7002	Method '**public** org.jdom.Document addContent(org.jdom.Element)' has been removed	**public**	Y	1:1	Rename	Delegated
org.jdom.Element:additionalNamespaces[LinkedList]	org.jdom.Element:additionalNamespaces[List]	6004	Changed type of field additionalNamespaces from java.util.LinkedList to java.util.List	**protected**	N	1:n	Change Type	Change from LinkedList to List, which removes some functionality.
org.jdom.Element:content[LinkedList]	org.jdom.Element:content[List]	6004	Changed type of field content from java.util.LinkedList to java.util.List	**protected**	N	1:n	Change Type	Change from LinkedList to List, which removes some functionality.

Continued on next page

Table E.8 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.Element:document[Document]	org.jdom.Element:parent[Object]	6001	Removed field document	**protected**	N	n:m	Remap; Change Type; Consolidate	Was keeping two variables to determine if it had a parent, or was the parent. Merged into a single Object, and uses reflection on the object's type to determine which is which now.
org.jdom.Element:parent[Element]	org.jdom.Element:parent[Object]	6004	Changed type of field parent from org.jdom.Element to java.lang.Object	**protected**	N	n:m	Change Type; Consolidate	Was keeping two variables to determine if it had a parent, or was the parent. Merged into a single Object, and uses reflection on the object's type to determine which is which now.
org.jdom.Element.removeContent(Entity)[**boolean**]	org.jdom.Element.removeContent(EntityRef)[**boolean**]	7005	Parameter 1 of '**public boolean** removeContent(org.jdom.Entity)' has changed its type to org.jdom.CDATA	**public**	N	1:n	Change Type	Entity renamed to EntityRef.
org.jdom.Element.addContent(Entity)[Element]	org.jdom.Element.addContent(EntityRef)[Element]	7005	Parameter 1 of '**public** org.jdom.Element addContent(org.jdom.Entity)' has changed its type to org.jdom.EntityRef	**public**	N	1:n	Change Type	
org.jdom.Entity	org.jdom.EntityRef	8001	Class org.jdom.Entity removed	**public**	N	1:n	Change Type	Renamed to EntityRef.

Continued on next page

Table E.8 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.IllegalAddException. IllegalAddException(Element, Entity,String)	org.jdom.IllegalAddException. IllegalAddException(Element, EntityRef,String)	7005	Parameter 2 of '**public** IllegalAddException(org. jdom.Element, org.jdom.Entity, java.lang.String)' has changed its type to org.jdom.EntityRef	**public**	N	1:n	Change Type	
org.jdom.JDOMException: rootCause[Throwable]	org.jdom.JDOMException:cause[Throwable]	6001	Removed field rootCause	**protected**	N	1:1	Rename	renamed
org.jdom.Namespace. getNamespace(String, Element)[Namespace]	org.jdom.Element.getNamespace (String)[Namespace]	7002	Method '**public** org.jdom. Namespace getNamespace(java. lang.String, org.jdom.Element)' has been removed	**public**	N	n:1	Consolidate	Code comment is misleading; it doesn't specify the class correctly, it uses an abbreviation.
none	org.jdom.adapters.DOMAdapter. createDocument(DocType)[Document]	7012	Method '**public** org.w3c.dom. Document createDocument(org. jdom.DocType)' has been added to an interface	**public**	N	1:n	Add method to interface	
org.jdom.output.XMLOutputter: padTextString[unknown]	none	6001	Removed field padTextString	**protected**	N	Delete	Delete	This was being used to pad spaces around Strings in XML tags, probably dropped as it's not needed—not to mention, causes hassles with data when reading.

Continued on next page

Table E.8 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.output.XMLOutputter.printCDATASection(CDATA, Writer, **int**)[**void**]	org.jdom.output.XMLOutputter.printCDATA(CDATA, Writer)[**void**]	7002	Method '**protected void** printCDATASection(org.jdom.CDATA, java.io.Writer, **int**)' has been removed	**protected**	N	1:1	Rename; Remove Parameter	
org.jdom.output.XMLOutputter.printComment(Comment, Writer, **int**)[**void**]	org.jdom.output.XMLOutputter.printComment(Comment, Writer)[**void**]	7004	In method '**protected void** printComment(org.jdom.Comment, java.io.Writer, **int**)' the number of arguments has changed	**protected**	N	1:1	Remove Parameter	
org.jdom.output.XMLOutputter.printEntity(Entity, Writer)[**void**]	org.jdom.output.XMLOutputter.printEntityRef(EntityRef, Writer)[**void**]	7002	Method '**protected void** printEntity(org.jdom.Entity, java.io.Writer)' has been removed	**protected**	N	1:n	Rename; Change Type	renamed
org.jdom.output.XMLOutputter.printNamespace(Namespace, Writer)[**void**]	org.jdom.output.XMLOutputter.printNamespace(Namespace, Writer)[**void**]	7009	Accessibility of method '**protected void** printNamespace(org.jdom.Namespace, java.io.Writer)' has been decreased from **protected** to **private**	**protected**	N	Delete	Hide	
org.jdom.output.XMLOutputter.printProcessingInstruction(ProcessingInstruction, Writer, **int**)[**void**]	org.jdom.output.XMLOutputter.printProcessingInstruction(ProcessingInstruction, Writer)[**void**]	7004	In method '**protected void** printProcessingInstruction(org.jdom.ProcessingInstruction, java.io.Writer, **int**)' the number of arguments has changed	**protected**	N	1:1	Remove Parameter	

Continued on next page

Table E.8 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.output.XMLOutputter.output(Entity, OutputStream)[**void**]	org.jdom.output.XMLOutputter.output(EntityRef, OutputStream)[**void**]	7005	Parameter 1 of '**public void** output(org.jdom.Entity, java.io.OutputStream)' has changed its type to org.jdom.DocType	**public**	N	1:1	Remove Parameter	
org.jdom.output.XMLOutputter.output(Entity, Writer)[**void**]	org.jdom.output.XMLOutputter.output(EntityRef, Writer)[**void**]	7005	Parameter 1 of '**public void** output(org.jdom.Entity, java.io.Writer)' has changed its type to org.jdom.DocType	**public**	N	1:1	Remove Parameter	

Table E.9: JDOM b7 to b8 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.Attribute:parent[Element]	org.jdom.Attribute:parent[Object]	6004	Changed type of field parent from org.jdom.Element to java.lang.Object	**protected**	N	1:n	Change Type	
org.jdom.Attribute. getSerializedForm()[String]	none	7002	Method '**public** java.lang.String getSerializedForm()' has been removed	**public**	Y	Delete	Delete	The code comment specifies a replacement that doesn't exist in the new version. You can replace this by manually specifying the replacement
org.jdom.CDATA:text[String]	org.jdom.CDATA:value[String]	6001	Removed field text	**protected**	N	1:1	Rename	renamed
org.jdom.CDATA. getSerializedForm()[String]	org.jdom.output.XMLOutputter. outputString(CDATA)[String]	7002	Method '**public** java.lang.String getSerializedForm()' has been removed	**public**	Y	1:n	Remap	Deprecation comment indicates replacement; note that a class instantiation is now also required.
org.jdom.Comment:document[Document]	org.jdom.Comment:parent[Object]	6001	Removed field document	**protected**	N	1:n	Remap; Change Type; Consolidate	Two fields are merged into one, and reflection is used to decide if the parent is a Document, or Element.
org.jdom.Comment:parent[Element]	org.jdom.Comment:parent[Object]	6004	Changed type of field parent from org.jdom.Element to java.lang.Object	**protected**	N	1:n	Change Type	
org.jdom.Comment. getSerializedForm()[String]	org.jdom.output.XMLOutputter. outputString(Comment)[String]	7002	Method '**public** java.lang.String getSerializedForm()' has been removed	**public**	N	1:n	Remap	Deprecation comment indicates replacement; note that a class instantiation is now also required.

Continued on next page

Table E.9 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.DocType. getSerializedForm()[String]	org.jdom.output.XMLOutputter. outputString(DocType)[String]	7002	Method '**public** java.lang.String getSerializedForm()' has been removed	**public**	N	1:n	Remap	Deprecation comment indicates replacement; note that a class instantiation is now also required.
org.jdom.Document:content[List]	org.jdom.Document:content[ContentList]	6004	Changed type of field content from java.util.List to org.jdom.ContentList	**protected**	N	1:n	Change Type	
org.jdom.Document. removeProcessingInstruction(String)[**boolean**]	org.jdom.Document.getContent()[List]	7002	Method '**public boolean** removeProcessingInstruction(java .lang.String)' has been removed	**public**	Y	1:n	Expose/Unpack Implementa- tion	Code comment directs here. Essentially, callers are asked to implement this functionality on their own.
org.jdom.Document. removeProcessingInstructions(String)[**boolean**]	org.jdom.Document.getContent()[List]	7002	Method '**public boolean** removeProcessingInstructions(java.lang.String)' has been removed	**public**	Y	1:n	Expose/Unpack Implementa- tion	Code comment directs here. Essentially, callers are asked to implement this functionality on their own.
org.jdom.Document. getSerializedForm()[String]	org.jdom.output.XMLOutputter. outputString(Document)[String]	7002	Method '**public** java.lang.String getSerializedForm()' has been removed	**public**	N	1:n	Remap	
org.jdom.Document. getMixedContent()[List]	org.jdom.Document.getContent()[List]	7002	Method '**public** java.util.List getMixedContent()' has been removed	**public**	Y	1:1	Rename	

Continued on next page

Table E.9 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.Document. getProcessingInstructions()[List]	org.jdom.Document.getContent()[List]	7002	Method '**public** java.util.List getProcessingInstructions()' has been removed	**public**	Y	1:n	Expose/Unpack Implementa- tion	Code comment directs here. Essentially, callers are asked to implement this functionality on their own.
org.jdom.Document. getProcessingInstructions(String)[List]	org.jdom.Document.getContent()[List]	7002	Method '**public** java.util.List getProcessingInstructions(java. lang.String)' has been removed	**public**	Y	1:n	Expose/Unpack Implementa- tion	Code comment directs here. Essentially, callers are asked to implement this functionality on their own.
org.jdom.Document. setMixedContent(List)[Document]	org.jdom.Document.setContent(List)[Document]	7002	Method '**public** org.jdom.Document setMixedContent(java.util.List)' has been removed	**public**	Y	1:1	Remap	
org.jdom.Document. setProcessingInstructions(List)[Document]	org.jdom.Document.getContent()[List]	7002	Method '**public** org.jdom.Document setProcessingInstructions(java. util.List)' has been removed	**public**	Y	1:n	Expose/Unpack Implementa- tion	No, not a typo. Code comment directs here. Essentially, callers are asked to implement this functionality on their own. Instead of setting the instructions, you call getContent() and add them manually.

Continued on next page

Table E.9 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.Document. getProcessingInstruction(String)[ProcessingInstruction]	org.jdom.Document.getContent()[List]	7002	Method '**public** org.jdom. ProcessingInstruction getProcessingInstruction(java. lang.String)' has been removed	**public**	Y	1:n	Expose/Unpack Implementa- tion	Code comment directs here. Essentially, callers are asked to implement this functionality on their own.
org.jdom.Element:attributes[List]	org.jdom.Element:attributes[AttributedList]	6004	Changed type of field attributes from java.util.List to org.jdom.AttributeList	**protected**	N	1:n	Change Type	
org.jdom.Element:content[List]	org.jdom.Element:content[Content]	6004	Changed type of field content from java.util.List to org.jdom.ContentList	**protected**	N	1:n	Change Type	
org.jdom.Element. hasMixedContent()[**boolean**]	deleted	7002	Method '**public boolean** hasMixedContent()' has been removed	**public**	Y	Delete	Delete	Comment indicates the method isn't useful, and so is being removed.
org.jdom.Element. removeAttribute(String, String)[**boolean**]	org.jdom.Element. removeAttribute(String, Namespace)[**boolean**]	7004	In method '**public boolean** removeAttribute(java.lang.String, java.lang.String)' the number of arguments has changed	**public**	Y	1:n	Change Type	
org.jdom.Element. getSerializedForm()[String]	org.jdom.output.XMLOutputter. outputString(Element)[String]	7002	Method '**public** java.lang.String getSerializedForm()' has been removed	**public**	N	1:n	Remap	

Continued on next page

Table E.9 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.Element. getMixedContent()[List]	org.jdom.Element:getContent[List]	7002	Method '**public** java.util.List getMixedContent()' has been removed	**public**	Y	1:1	Rename	
org.jdom.Element.addAttribute(String, String)[Element]	org.jdom.Element.setAttribute(Attribute)[Element]	7002	Method '**public** org.jdom. Element addAttribute(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Rename; Remove Parameter; Change Type	
org.jdom.Element.addAttribute(Attribute)[Element]	org.jdom.Element.setAttribute(Attribute)[Element]	7002	Method '**public** org.jdom.Element addAttribute(org.jdom.Attribute)' has been removed	**public**	Y	1:1	Rename	
org.jdom.Element.getCopy(String)[Element]	org.jdom.Element.clone()[Element]; org.jdom.Element. setName(String)[Element]	7002	Method '**public** org.jdom.Element getCopy(java.lang.String)' has been removed	**public**	Y	1:n	Expose/Unpack Implementa- tion	Deprecation comment indicates replacement.
org.jdom.Element.getCopy(String, Namespace)[Element]	org.jdom.Element.clone()[Element]; org.jdom.Element. setName(String)[Element]	7002	Method '**public** org.jdom. Element getCopy(java.lang.String , org.jdom.Namespace)' has been removed	**public**	Y	1:n	Expose/Unpack Implementa- tion	Deprecation comment indicates replacement.
org.jdom.Element. setMixedContent(List)[Element]	org.jdom.Element.setContent(List)[Element]	7002	Method '**public** org.jdom.Element setMixedContent(java.util.List)' has been removed	**public**	Y	1:n	Remap; Change Type; Consolidate	

Continued on next page

Table E.9 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.EntityRef:document[Document]	org.jdom.EntityRef:parent[Object]	6001	Removed field document	**protected**	N	1:n	Change Type	
org.jdom.EntityRef:parent[Element]	org.jdom.EntityRef:parent[Object]	6004	Changed type of field parent from org.jdom.Element to java.lang.Object	**protected**	N	1:n	Change Type	
org.jdom.JDOMException.getRootCause()[Throwable]	org.jdom.JDOMException.getCause()[Throwable]	7002	Method '**public** java.lang.Throwable getRootCause()' has been removed	**public**	Y	1:1	Rename	Deprecation comment indicates replacement.
org.jdom.ProcessingInstruction:document[Document]	org.jdom.ProcessingInstruction:parent[Object]	6001	Removed field document	**protected**	N	1:n	Remap; Change Type; Consolidate	use instead based on context
org.jdom.ProcessingInstruction:parent[Element]	org.jdom.ProcessingInstruction:parent[Object]	6004	Changed type of field parent from org.jdom.Element to java.lang.Object	**protected**	N	1:n	Change Type	
org.jdom.ProcessingInstruction.getSerializedForm()[String]	org.jdom.output.XMLOutputter.outputString(ProcessingInstruction)[String]	7002	Method '**public** java.lang.String getSerializedForm()' has been removed	**public**	N	1:n	Remap	Deprecation comment indicates replacement; note that a class instantiation is now also required.
org.jdom.Text:parent[Document]	org.jdom.Text:parent[Object]	6004	Changed type of field parent from org.jdom.Element to java.lang.Object	**protected**	N	1:n	Change Type	

Continued on next page

Table E.9 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.Text:value[StringBuffer]	org.jdom.Text:value[String]	6004	Changed type of field value from java.lang.StringBuffer to java.lang.String	**protected**	N	1:n	Change Type	
org.jdom.Text.getValue()[String]	org.jdom.Text.getText()[String]	7002	Method '**public** java.lang.String getValue()' has been removed	**public**	N	1:1	Rename	
org.jdom.Text.setValue(String)[**void**]	org.jdom.Text.setText(String)[**void**]	7002	Method '**public void** setValue(java.lang.String)' has been removed	**public**	N	1:1	Rename	renamed
none	org.jdom.input.JDOMFactory.attribute(String, String, **int**)[Attribute]	7012	Method '**public** org.jdom.Attribute attribute(java.lang.String, java.lang.String, **int**)' has been added to an interface	**public**	N	1:n	Add method to interface	
none	org.jdom.input.JDOMFactory.attribute(String, String, **int**, Namespace)[Attribute]	7012	Method '**public** org.jdom.Attribute attribute(java.lang.String, java.lang.String, **int**, org.jdom.Namespace)' has been added to an interface	**public**	N	1:n	Add method to interface	
none	org.jdom.input.JDOMFactory.text(String)[Text]	7012	Method '**public** org.jdom.Text text(java.lang.String)' has been added to an interface	**public**	N	1:n	Add method to interface	

Continued on next page

Table E.9 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.output.XMLOutputter: STANDARD_INDENT[String]	org.jdom.output.XMLOutputter: STANDARD_INDENT[String]; org.jdom.output.XMLOutputter.setIndent(**boolean**)[**void**]	6010	Accessibility of field STANDARD_INDENT has been weakened from **protected** to **private**	**protected**	N	1:1	Hide; Encapsulate (Partial)	
org.jdom.output.XMLOutputter.maybePrintln(Writer)[**void**]	org.jdom.output.XMLOutputter.newline(Writer)[**void**]	7002	Method '**protected void** maybePrintln(java.io.Writer)' has been removed	**protected**	N	1:1	Rename	
org.jdom.output.XMLOutputter.printElementContent(Element, Writer, **int**, NamespaceStack, List)[**void**]	org.jdom.output.XMLOutputter.printElementContent(Element, Writer, **int**, NamespaceStack)[**void**]	7004	In method '**protected void** printElementContent(org.jdom.Element, java.io.Writer, **int**, org.jdom.output.XMLOutputter\$NamespaceStack, java.util.List)' the number of arguments has changed	**protected**	N	1:1	Remove Parameter	
org.jdom.output.XMLOutputter.setTrimText(**boolean**)[**void**]	org.jdom.output.XMLOutputter.setTextNormalize(**boolean**)[**void**]	7002	Method '**public void** setTrimText(**boolean**)' has been removed	**public**	Y	1:1	Rename	

Table E.10: JDOM b8 to b9 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.CDATA:parent[Object]	org.jdom.Text:parent[Object]	6001	Removed field parent	**protected**	N	1:1	Pull Up	
org.jdom.CDATA:value[String]	org.jdom.Text:value[String]	6001	Removed field value	**protected**	N	1:1	Pull Up	
org.jdom.CDATA.setParent(Element)[CDATA]	org.jdom.Text.setParent(Element)[Text]	7002	Method '**protected** org.jdom.CDATA setParent(org.jdom.Element)' has been removed	**protected**	N	1:1	Pull Up; Change Type	
org.jdom.CDATA.detach()[CDATA]	org.jdom.Text.detach()[Text]	7002	Method '**public** org.jdom.CDATA detach()' has been removed	**public**	N	1:1	Pull Up; Change Type	
org.jdom.CDATA.setText(String)[CDATA]	org.jdom.CDATA.setText(String)[Text]	7006	Return type of method '**public** org.jdom.CDATA setText(java.lang.String)' has been changed to org.jdom.Text	**public**	N	1:1	Pull Up; Change Type	
org.jdom.CDATA.append(CDATA)[**void**]	org.jdom.Text.append(Text)[**void**]	7002	Method '**public void** append(org.jdom.CDATA)' has been removed	**public**	N	1:1	Pull Up; Change Type	
org.jdom.Element.removeContent(CDATA)[**boolean**]	org.jdom.Element.removeContent(Text)[**boolean**]	7002	Method '**public boolean** removeContent(org.jdom.CDATA)' has been removed	**public**	N	1:1	Change Type	

Continued on next page

Table E.10 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.Element.addContent(CDATA)[Element]	org.jdom.Element.addContent(Text)[Element]	7002	Method '**public** org.jdom.Element addContent(org.jdom.CDATA)' has been removed	**public**	N	1:1	Change Type	
org.jdom.Verifier.main(String[])[**void**]	none	7002	Method '**public void** main(java. lang.String[])' has been removed	**public**	N	Delete	Delete	The main() method was meant to be a test harness, looks like it's removed now. Couldn't find a replacement unit test.
org.jdom.adapters. ProjectXDOMAdapter	none	8001	Class org.jdom.adapters.ProjectXDOMAdapter removed	**public**	N	Delete	Delete	This is an adaptor for Sun's "Project X"; I can't figure out what that could be, but it looks like support for that was dropped.
org.jdom.filter.ContentFilter. canAdd(Object)[**boolean**]	org.jdom.filter.ContentFilter. matches(Object)[**boolean**]	7002	Method '**public boolean** canAdd(java.lang.Object)' has been removed	**public**	N	1:1	Rename	Delegated
org.jdom.filter.ContentFilter. canRemove(Object)[**boolean**]	org.jdom.filter.ContentFilter. matches(Object)[**boolean**]	7002	Method '**public boolean** canRemove(java.lang.Object)' has been removed	**public**	N	1:1	Rename	Delegated
org.jdom.filter.ElementFilter. canAdd(Object)[**boolean**]	org.jdom.filter.ElementFilter. matches(Object)[**boolean**]	7002	Method '**public boolean** canAdd(java.lang.Object)' has been removed	**public**	N	1:1	Rename	Delegated

Continued on next page

Table E.10 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.filter.ElementFilter.canRemove(Object)[**boolean**]	org.jdom.filter.ElementFilter.matches(Object)[**boolean**]	7002	Method '**public boolean** canRemove(java.lang.Object)' has been removed	**public**	N	1:1	Rename	Delegated
org.jdom.filter.Filter.canAdd(Object)[**boolean**]	org.jdom.filter.Filter.matches(Object)[**boolean**]	7002	Method '**public boolean** canAdd(java.lang.Object)' has been removed	**public**	N	1:1	Rename	
org.jdom.filter.Filter.canRemove(Object)[**boolean**]	org.jdom.filter.Filter.matches(Object)[**boolean**]	7002	Method '**public boolean** canRemove(java.lang.Object)' has been removed	**public**	N	1:1	Rename	
org.jdom.input.SAXHandler: availableNamespaces[LinkedList]	none	6001	Removed field availableNamespaces	**protected**	N	Delete	Deleted	Meant to track Namespaces actually in use or usable, I can't determine what the replacement functionality could be, it just seems to be unnecessary now
org.jdom.input.SAXHandler: declaredNamespaces[LinkedList]	org.jdom.input.SAXHandler: declaredNamespaces[List]	6004	Changed type of field declaredNamespaces from java.util.LinkedList to java.util.List	**protected**	N	1:n	Change Type	
org.jdom.input.SAXHandler:stack[Stack]	org.jdom.input.SAXHandler: atRoot[**boolean**]	6001	Removed field stack	**protected**	N	1:n	Change Type; Expose/Unpack Implementation	Radical type change: the Stack was used to see whether you were at the root of the tree or not. It can be done now with this field.

Continued on next page

Table E.10 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.input.SAXHandler. SAXHandler(Document)	org.jdom.input.SAXHandler. SAXHandler()	7002	Method '**public** SAXHandler(org.jdom.Document)' has been removed	**public**	N	1:1	Remove Parameter	
org.jdom.output.XMLOutputter. indent(Writer)[**void**]	org.jdom.output.XMLOutputter. indent(Writer,**int**)[**void**]	7002	Method '**protected void** indent (java.io.Writer)' has been removed	**protected**	Y	1:1	Add contextual data	By hardcoding a parameter to 0, this method is equivalent.
org.jdom.output.XMLOutputter. printContent(List, Writer, **int**, NamespaceStack)[**void**]	org.jdom.output.XMLOutputter. printContentRange(List,**int**,**int**, Writer, **int**, NamespaceStack)[**void**]	7002	Method '**protected void** printContent(java.util.List, java.io.Writer, **int**, org.jdom.output.XMLOutputter\ $NamespaceStack)' has been removed	**protected**	N	1:1	Add contextual data	pass 0, content.size() for 2nd and 3rd parameters
org.jdom.output.XMLOutputter. printElementContent(Element, Writer, **int**, XMLOutputter\ $NamespaceStack)[**void**]	org.jdom.output.XMLOutputter. printContentRange(List,**int**,**int**, Writer, **int**, XMLOutputter\ $NamespaceStack)[**void**]	7002	Method '**protected void** printElementContent(org.jdom. Element, java.io.Writer, **int**, org. jdom.output.XMLOutputter\ $NamespaceStack)' has been removed	**protected**	N	1:1	Add contextual data	pass element.getContent, 0, content.size() for 1st, 2nd and 3rd parameters

Continued on next page

Table E.10 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.jdom.output.XMLOutputter. setIndentLevel(**int**)[**void**]	java.io.FilterOutputStream	7002	Method '**public void** setIndentLevel(**int**)' has been removed	**public**	Y	1:n	Remap; Change Type; Extract to External API; Expose/Unpack Implementation	The comment in the code says to stack FilterOutputStreams to do the indentation. I think that what they mean is to apply successive layers of FOS's, one for each level of indentation, and then apply a manual indent to each one.
org.jdom.output.XMLOutputter. setPadText(**boolean**)[**void**]	deleted	7002	Method '**public void** setPadText(**boolean**)' has been removed	**public**	N	Delete	Deleted	Comment in the code says this is no longer necessary.
org.jdom.output.XMLOutputter. setSuppressDeclaration(**boolean**)[**void**]	org.jdom.output.XMLOutputter. setOmitDeclaration(**boolean**)[**void**]	7002	Method '**public void** setSuppressDeclaration(**boolean**)' has been removed	**public**	Y	1:1	Rename	

Table E.11: LOG4J 1.0.4 to 1.1.3 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
none	org.apache.log4j.Appender.getLayout()[Layout]	7012	Method '**public** org.apache.log4j.Layout getLayout()' has been added to an interface	**public**	N	1:n	Add to Interface	Just a getter.
none	org.apache.log4j.Appender.getErrorHandler()[ErrorHandler]	7012	Method '**public** org.apache.log4j.spi.ErrorHandler getErrorHandler()' has been added to an interface	**public**	N	1:n	Add to Interface	Abstract class AppenderSkeleton had an implementation of this previously, method got promoted to interface. Just a getter
none	org.apache.log4j.Appender.getFilter()[Filter]	7012	Method '**public** org.apache.log4j.spi.Filter getFilter()' has been added to an interface	**public**	N	1:n	Add to Interface	Abstract class AppenderSkeleton's implementation of this shows it to be equivalent as a getter to getFirstFilter().
org.apache.log4j.BasicConfigurator.addRenderer(String, String)[**void**]	org.apache.log4j.BasicConfigurator.addRenderer(Hierarchy, String, String)	7004	In method '**protected void** addRenderer(java.lang.String, java.lang.String)' the number of arguments has changed	**protected**	N	1:n	Add contextual data	
org.apache.log4j.BasicConfigurator.overrideAsNeeded(String)[**void**]	org.apache.log4j.Hierarchy.overrideAsNeeded(String)[**void**]	7002	Method '**protected void** overrideAsNeeded(java.lang.String)' has been removed	**protected**	N	1:1	Move	Moved
org.apache.log4j.BasicConfigurator.flagAsShippedCode()[**void**]	org.apache.log4j.BasicConfigurator.disableInfo()[**void**]	7002	Method '**public void** flagAsShippedCode()' has been removed	**public**	Y	1:1	Rename	Delegated

Continued on next page

Table E.11 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.Category: disable[**int**]	org.apache.log4j.Hierarchy: disable[**int**]	6001	Removed field disable	**protected**	N	1:1	Move	
org.apache.log4j.Category: instanceFQN[String]	Class.getFullyQualified()[String]	6001	Removed field instanceFQN	**protected**	N	1:1	Extract to External API	
org.apache.log4j.Category: myContext[Hierarchy]	org.apache.log4j.Category: hierarchy[Hierarchy]	6001	Removed field myContext	**protected**	N	1:1	Rename	
org.apache.log4j.FileAppender: immediateFlush[**boolean**]	org.apache.log4j.WriterAppender: immediateFlush[**boolean**]	6001	Removed field immediateFlush	**protected**	N	1:n	Pull Up; Supertype Change	Now relies on JAVABEANS introspection to set/get options like this.
org.apache.log4j.FileAppender: qw[QuietWriter]	org.apache.log4j.WriterAppender: qw[QuietWriter]	6001	Removed field qw	**protected**	N	1:n	Pull Up; Supertype Change	
org.apache.log4j.FileAppender:tp[QuietWriter]	org.apache.log4j.WriterAppender: qw[QuietWriter]	6001	Removed field tp	**protected**	N	1:n	Remap; Expose/Unpack Implementation	was basically a duplicate of the QuietWriter, WriterAppender.subAppend shows the additional code to make the output of QW equivalent to TP.
org.apache.log4j.FileAppender. closeWriterIfOurs()[**void**]	org.apache.log4j.FileAppender. closeFile()[**void**]	7002	Method '**protected void** closeWriterIfOurs()' has been removed	**protected**	N	1:1	Remap	

Continued on next page

Table E.11 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.RollingFileAppender.RollingFileAppender(Layout, OutputStream)	org.apache.log4j.RollingFileAppender.RollingFileAppender(Layout, String)	7002	Method '**public** RollingFileAppender(org.apache.log4j.Layout, java.io.OutputStream)' has been removed	**public**	Y	1:n	Change Type	
org.apache.log4j.RollingFileAppender.RollingFileAppender(Layout, Writer)	org.apache.log4j.RollingFileAppender.RollingFileAppender(Layout, String)	7002	Method '**public** RollingFileAppender(org.apache.log4j.Layout, java.io.Writer)' has been removed	**public**	Y	1:n	Change Type	
org.apache.log4j.gui.TextPaneAppender	deleted	8001	Class org.apache.log4j.gui.TextPaneAppender removed	**public**	N	Delete	Delete	Looks like this was a GUI frontend that got dropped; code comment indicated it was experimental.
org.apache.log4j.helpers.Loader.getGIF_Image(String)[Image]	org.apache.log4j.helpers.Loader.getResource(String, Class)[URL]; java.awt.Toolkit.getImage(URL)[Image]	7002	Method '**public** java.awt.Image getGIF_Image(java.lang.String)' has been removed	**public**	N	1:n	Expose/Unpack Implementation	
org.apache.log4j.helpers.Loader.getGIF_Image(URL)[Image]	java.awt.Toolkit.getImage(URL)[Image]	7002	Method '**public** java.awt.Image getGIF_Image(java.net.URL)' has been removed	**public**	N	1:1	Expose/Unpack Implementation	

Continued on next page

Table E.11 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.helpers.Loader. getHTML_Page(String)[URL]	org.apache.log4j.helpers.Loader. getResource(String, Class)[URL]	7002	Method '**public** java.net.URL getHTML_Page(java.lang.String)' has been removed	**public**	N	1:n	Rename; Add Contextual Data	
org.apache.log4j.helpers.LogLog: configDebugEnabled[**boolean**]	org.apache.log4j.helpers.LogLog. debugEnabled[**boolean**]	6001	Removed field configDebugEnabled	**protected**	N	1:1	Rename	
org.apache.log4j.helpers. TracerPrintWriter	org.apache.log4j.helpers. QuietWriter	8001	Class org.apache.log4j.helpers. TracerPrintWriter removed	**public**	N	1:n	Change Type	TracerPrinter was just a wrapper. Functionality requires some work to remap. This is a good example of a 1:n replacement.
org.apache.log4j.net. JMSAppender: TOPIC_BINDING_NAME_OPTION [String]	org.apache.log4j.net. JMSAppender: TOPIC_BINDING_NAME_OPTION [String]	6006	Field TOPIC_BINDING_NAME_OPTION is now **final**	**protected**	N	1:n	Encapsulate (Partial); Immutable	
org.apache.log4j.net. JMSAppender: TOPIC_CONNECTION_FACTORY_BINDING_NAME_OPTION [String]	org.apache.log4j.net. JMSAppender: TOPIC_CONNECTION_FACTORY_BINDING_NAME_OPTION [String]	6006	Field TOPIC_CONNECTION_FACTORY_BINDING_NAME_OPTION is now **final**	**protected**	N	1:n	Encapsulate (Partial); Immutable	
org.apache.log4j.spi. LoggingEvent:categoryName[String]	org.apache.log4j.spi. LoggingEvent:categoryName[String]	6006	Field categoryName is now **final**	**public**	N	1:1	Immutable	

Continued on next page

Table E.11 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.spi.LoggingEvent:fqnOfCategoryClass[String]	org.apache.log4j.spi.LoggingEvent:fqnOfCategoryClass[String]	6006	Field fqnOfCategoryClass is now **final**	**public**	N	1:1	Immutable	
org.apache.log4j.spi.LoggingEvent:locationInfo[LocationInfo]	org.apache.log4j.spi.LoggingEvent.getLocationInformation()[LocationInfo]	6010	Accessibility of field locationInfo has been weakened from **public** to **private**	**public**	N	1:n	Encapsulate (Partial); Hide	
org.apache.log4j.spi.LoggingEvent:message[String]	org.apache.log4j.spi.LoggingEvent.getRenderedMessage()[String]	6004	Changed type of field message from java.lang.String to java.lang.Object	**public**	N	1:1	Change Type; Encapsulate (Partial)	**public** to **private**, field type changed from String to Object. There's a couple of appropriate getters, this is the best.
org.apache.log4j.spi.LoggingEvent:message[String]	org.apache.log4j.spi.LoggingEvent.getRenderedMessage()[String]	6010	Accessibility of field message has been weakened from **public** to **private**	**public**	N	1:1	Hide; Encapsulate (Partial)	Note that multiple BIs are reported for this change; see the previous.
org.apache.log4j.spi.LoggingEvent:ndc[String]	org.apache.log4j.spi.LoggingEvent.getNDC()[String]	6010	Accessibility of field ndc has been weakened from **public** to **private**	**public**	N	1:1	Hide; Encapsulate (Partial)	**public** to **private**, only a getter offered now.
org.apache.log4j.spi.LoggingEvent:ndcLookupRequired[**boolean**]	org.apache.log4j.spi.LoggingEvent:ndcLookupRequired[**boolean**]	6010	Accessibility of field ndcLookupRequired has been weakened from **public** to **private**	**public**	N	1:1	Hide	**public** to **private**, no replacement

Continued on next page

Table E.11 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.spi.LoggingEvent:threadName[String]	org.apache.log4j.spi.LoggingEvent.getThreadName()[String]	6010	Accessibility of field threadName has been weakened from **public** to **private**	**public**	N	1:1	Hide; Encapsulate (Partial)	**public** to **private**, only a getter offered now.
org.apache.log4j.spi.LoggingEvent:throwable[Throwable]	org.apache.log4j.spi.LoggingEvent:throwableInfo[ThrowableInformation]	6001	Removed field throwable	**public**	N	1:n	Hide; Encapsulate (Partial); Change Type	
org.apache.log4j.spi.LoggingEvent:throwableInformation[String]	org.apache.log4j.spi.LoggingEvent.getThrowableStrRep()[String[]]	6001	Removed field throwableInformation	**public**	N	1:1	Change Type; Hide; Encapsulate (Partial)	field deleted, this method provides same functionality except that it provides an array of strings, rather than a single concatenated one. Could also treat as a 1:n replacement.
org.apache.log4j.spi.LoggingEvent:timeStamp[**long**]	org.apache.log4j.spi.LoggingEvent:timeStamp[**long**]	6006	Field timeStamp is now **final**	**public**	N	1:1	Immutable	**final**
org.apache.log4j.spi.LoggingEvent.LoggingEvent(String, Category, Priority, String, Throwable)	org.apache.log4j.spi.LoggingEvent.LoggingEvent(String, Category, Priority, Object, Throwable)	7005	Parameter 4 of '**public** LoggingEvent(java.lang.String, org.apache.log4j.Category, org.apache.log4j.Priority, java.lang.String, java.lang.Throwable)' has changed its type to java.lang.Object	**public**	N	1:1	Change Type	

Continued on next page

Table E.11 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.spi.LoggingEvent.getThrowableInformation()[String]	org.apache.log4j.spi.LoggingEvent.getThrowableInformation()[ThrowableInformation]	7006	Return type of method '**public** java.lang.String getThrowableInformation()' has been changed to org.apache.log4j.spi.ThrowableInformation	**public**	N	1:n	Change Type	
org.apache.log4j.spi.LoggingEvent.setLocationInformation()[**void**]	org.apache.log4j.spi.LoggingEvent.getLocationInformation()[LocationInfo]	7002	Method '**public void** setLocationInformation()' has been removed	**public**	N	1:1	Rename	amazing, but true: the old set function just instantiated a default value. The new get function does the same thing, then returns a copy.
org.apache.log4j.varia.ResilientFileAppender	org.apache.log4j.varia.ExternallyRolledFileAppender	8001	Class org.apache.log4j.varia.ResilientFileAppender removed	**public**	N	1:n	Remap	Documentation indicates that the recommended alternative is org.apache.log4j.varia.ExternallyRolledFileAppender, but these are very different classes and would require significant adaptation work. I'm not sure how to do it just from looking at the code. Potentially a 1:n.
org.apache.log4j.xml.DOMConfigurator.parseParameters(Element, OptionHandler)[**void**]	org.apache.log4j.xml.DOMConfigurator.setParameter(Element, PropertySetter)[**void**]	7002	Method '**protected void** parseParameters(org.w3c.dom.Element, org.apache.log4j.spi.OptionHandler)' has been removed	**protected**	N	1:1	Rename; Change Type	

Continued on next page

Table E.11 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.xml. DOMConfigurator. parserRenderer(Element)[**void**]	org.apache.log4j.xml. DOMConfigurator. parserRenderer(Element, Hierarchy)[**void**]	7002	Method '**protected void** parserRenderer(org.w3c.dom.Element)' has been removed	**protected**	N	1:n	Add contextual data	

Table E.12: LOG4J 1.1.3 to 1.2.1 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j. AppenderSkeleton. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j. AppenderSkeleton.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.AsyncAppender. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.AsyncAppender. setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j. BasicConfigurator.addRenderer(Hierarchy, String, String)[**void**]	org.apache.log4j.or.RendererMap .addRenderer(LoggerRepository, String, String)[**void**]	7002	Method '**protected void** addRenderer(org.apache.log4j.Hierarchy, java. lang.String, java.lang.String)' has been removed	**protected**	N	1:n	Move; Change Type	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.BasicConfigurator.disable(String)[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(String)[**void**]	7002	Method '**public void** disable(java.lang.String)' has been removed	**public**	Y	1:n	Remap	Not clear if the original was used to selectively activate or deactivate each logging error level/priority, but the replacement is to set a logging threshold, and by default everything more important than that level gets logged.
org.apache.log4j.BasicConfigurator.disable(Priority)[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level)[**void**]	7002	Method '**public void** disable(org.apache.log4j.Priority)' has been removed	**public**	Y	1:n	Remap; Change Type	Not clear if the original was used to selectively activate or deactivate each logging error level/priority, but the replacement is to set a logging threshold, and by default everything more important than that level gets logged.
org.apache.log4j.BasicConfigurator.disableAll()[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level.OFF)[**void**]	7002	Method '**public void** disableAll()' has been removed	**public**	Y	1:1	Remap; Add Contextual Data	Note that this may behave slightly differently; previously, FATAL was the highest logging level, OFF didn't exist. So I'm not sure that the original disableAll() actually turned off FATAL logging or not.

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.BasicConfigurator.disableDebug()[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level.DEBUG)[**void**]	7002	Method '**public void** disableDebug()' has been removed	**public**	Y	1:1	Remap; Add Contextual Data	
org.apache.log4j.BasicConfigurator.disableInfo()[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level.INFO)[**void**]	7002	Method '**public void** disableInfo()' has been removed	**public**	Y	1:1	Remap; Add Contextual Data	
org.apache.log4j.BasicConfigurator.enableAll()[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level.ALL)[**void**]	7002	Method '**public void** enableAll()' has been removed	**public**	Y	1:1	Remap; Add Contextual Data	
org.apache.log4j.BasicConfigurator.resetConfiguration(Hierarchy)[**void**]	org.apache.log4j.Hierarchy.resetConfiguration()[**void**]	7002	Method '**public void** resetConfiguration(org.apache.log4j.Hierarchy)' has been removed	**public**	Y	1:1	Expose/Unpack Implementation	
org.apache.log4j.Category:defaultHierarchy[Hierarchy]	org.apache.log4j.LogManager.getLoggerRepository()[LoggerRepository]	6001	Removed field defaultHierarchy	**public**	N	1:n	Expose/Unpack Implementation; Change Type	
org.apache.log4j.Category:hierarchy[Hierarchy]	org.apache.log4j.Category:repository[LoggerRepository]	6001	Removed field hierarchy	**protected**	N	1:n	Rename; Change Type	
org.apache.log4j.Category:priority[Priority]	org.apache.log4j.Category:level[Level]	6001	Removed field priority	**protected**	N	1:1	Rename; Change Type	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.Category.exists(String)[Category]	org.apache.log4j.spi.LoggerRepository.exists(String)[Logger]	7006	Return type of method '**public** org.apache.log4j.Category exists(java.lang.String)' has been changed to org.apache.log4j.Logger	**public**	N	1:1	Change Type	
org.apache.log4j.Category.getInstance(String, CategoryFactory)[Category]	org.apache.log4j.spi.LoggerRepository.getLogger(String, LoggerFactory)[Logger]	7002	Method '**public** org.apache.log4j.Category getInstance(java.lang.String, org.apache.log4j.spi.CategoryFactory)' has been removed	**public**	N	1:n	Remap; Change Type	
org.apache.log4j.Category.getDefaultHierarchy()[Hierarchy]	org.apache.log4j.LogManager.getLoggerRepository()[LoggerRepository]	7006	Return type of method '**public** org.apache.log4j.Hierarchy getDefaultHierarchy()' has been changed to org.apache.log4j.spi.LoggerRepository	**public**	N	1:n	Expose/Unpack Implementation; Change Type	
org.apache.log4j.Category.getHierarchy()[Hierarchy]	org.apache.log4j.Category.getLoggerRepository[LoggerRepository]	7006	Return type of method '**public** org.apache.log4j.Hierarchy getHierarchy()' has been changed to org.apache.log4j.spi.LoggerRepository	**public**	N	1:n	Rename; Change Type	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.Category. getPriority()[Priority]	org.apache.log4j.Category. getLevel[Level]	7006	Return type of method '**public** org.apache.log4j.Priority getPriority()' has been changed to org.apache.log4j.Level	**public**	N	1:1	Rename; Change Type	
org.apache.log4j.Category.assert (**boolean**, String)[**void**]	org.apache.log4j.Category. assertLog(**boolean**, String)[**void**]	7002	Method '**public void** assert(**boolean**, java.lang.String)' has been removed	**public**	N	1:1	Rename	
org.apache.log4j. ConsoleAppender. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j. ConsoleAppender.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j. DailyRollingFileAppender. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j. DailyRollingFileAppender. setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.FileAppender: qwIsOurs[**boolean**]	none	6001	Removed field qwIsOurs	**protected**	Y	Delete	Delete	Comment specifically indicates it's being removed as FileAppenders are now required to "own" their own QWs.
org.apache.log4j.FileAppender. FileAppender(Layout, OutputStream)	org.apache.log4j.WriterAppender. WriterAppender(Layout, OutputStream)	7004	In method '**public** FileAppender(org.apache.log4j. Layout, java.io.OutputStream)' the number of arguments has changed	**public**	Y	1:1	Move	
org.apache.log4j.FileAppender. FileAppender(Layout, Writer)	org.apache.log4j.WriterAppender. WriterAppender(Layout, Writer)	7002	Method '**public** FileAppender(org.apache .log4j.Layout, java.io.Writer)' has been removed	**public**	Y	1:1	Move	
org.apache.log4j.FileAppender. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.FileAppender. setFile(String, **boolean**)[**void**]	org.apache.log4j.FileAppender. setFile(String, **boolean**, **boolean**, **int**)[**void**]	7004	In method '**public void** setFile(java.lang.String, **boolean**)' the number of arguments has changed	**public**	N	1:n	Add Contextual Data	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.FileAppender. setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.HTMLLayout. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.HTMLLayout. setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.Hierarchy. Hierarchy(Category)	org.apache.log4j.Hierarchy. Hierarchy(Logger)	7005	Parameter 1 of '**public** Hierarchy(org.apache. log4j.Category)' has changed its type to org.apache.log4j.Logger	**public**	N	1:n	Change Type	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.Hierarchy.exists(String)[Category]	org.apache.log4j.spi.LoggerRepository.exists(String)[Logger]	7006	Return type of method 'public org.apache.log4j.Category exists(java.lang.String)' has been changed to org.apache.log4j.Logger	public	N	1:1	Change Type	Note that there is a replacement also in the new version of Hierarchy that's also a Change Type 1:1 transformation. I chose this one instead because it's (a) the recommended means of doing it, and (b) other calls are also being updated to this.
org.apache.log4j.Hierarchy.getInstance(String)[Category]	org.apache.log4j.LogManager.getLogger(String)[Logger]	7002	Method 'public org.apache.log4j.Category getInstance(java.lang.String)' has been removed	public	N	1:n	Move; Rename; Change Type	
org.apache.log4j.Hierarchy.getInstance(String, CategoryFactory)[Category]	org.apache.log4j.spi.LoggerRepository.getLogger(String, LoggerFactory)[Logger]	7002	Method 'public org.apache.log4j.Category getInstance(java.lang.String, org.apache.log4j.spi.CategoryFactory)' has been removed	public	N	1:n	Move; Rename; Change Type	
org.apache.log4j.Hierarchy.getRoot()[Category]	org.apache.log4j.Hierarchy.getRootLogger()[Logger]	7002	Method 'public org.apache.log4j.Category getRoot()' has been removed	public	N	1:n	Rename; Change Type	
org.apache.log4j.Hierarchy.disable(String)[void]	org.apache.log4j.spi.LoggerRepository.setThreshold(String)[void]	7002	Method 'public void disable(java.lang.String)' has been removed	public	N	1:n	Remap	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.Hierarchy.disable(Priority)[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level)[**void**]	7002	Method '**public void** disable(org.apache.log4j.Priority)' has been removed	**public**	N	1:n	Remap; Change Type	
org.apache.log4j.Hierarchy.disableAll()[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level.OFF)[**void**]	7002	Method '**public void** disableAll()' has been removed	**public**	N	1:1	Remap; Add Contextual Data	
org.apache.log4j.Hierarchy.disableDebug()[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level.DEBUG)[**void**]	7002	Method '**public void** disableDebug()' has been removed	**public**	N	1:1	Remap; Add Contextual Data	
org.apache.log4j.Hierarchy.disableInfo()[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level.INFO)[**void**]	7002	Method '**public void** disableInfo()' has been removed	**public**	N	1:1	Remap; Add Contextual Data	
org.apache.log4j.Hierarchy.enableAll()[**void**]	org.apache.log4j.spi.LoggerRepository.setThreshold(Level.ALL)[**void**]	7002	Method '**public void** enableAll()' has been removed	**public**	N	1:1	Remap; Add Contextual Data	
org.apache.log4j.Hierarchy.setCategoryFactory(CategoryFactory)[**void**]	org.apache.log4j.Hierarchy.Hierarchy(Logger)[**void**]	7002	Method '**public void** setCategoryFactory(org.apache.log4j.spi.CategoryFactory)' has been removed	**public**	N	n:1	Consolidate	The field is now set at class creation with a default factory, and there's no other way to set it, so basically this method got rolled into the constructor
org.apache.log4j.Layout.getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String[] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.Layout. setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.PatternLayout. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.PatternLayout. setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j. PropertyConfigurator	org.apache.log4j. PropertyConfigurator	5001	Removed org.apache.log4j.BasicConfigurator from the list of superclasses	**public**	N	1:n	Supertype Change	
org.apache.log4j. PropertyConfigurator: categoryFactory[CategoryFactory]	org.apache.log4j. PropertyConfigurator: loggerFactory[LoggerFactory]	6001	Removed field categoryFactory	**protected**	N	1:n	Rename; Change Type	
org.apache.log4j. PropertyConfigurator. configureCategoryFactory(Properties)[**void**]	org.apache.log4j. PropertyConfigurator. configureLoggerFactory(Properties)[**void**]	7002	Method '**protected void** configureCategoryFactory(java. util.Properties)' has been removed	**protected**	N	1:1	Rename	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j. PropertyConfigurator. parseCatsAndRenderers(Properties, Hierarchy)[**void**]	org.apache.log4j. PropertyConfigurator. parseCatsAndRenderers(Properties, LoggerRepository)[**void**]	7005	Parameter 2 of '**protected void** parseCatsAndRenderers(java.util.Properties, org.apache. log4j.Hierarchy)' has changed its type to org.apache.log4j.spi. LoggerRepository	**public**	N	1:n	Change Type	
org.apache.log4j. PropertyConfigurator.doConfigure (String, Hierarchy)[**void**]	org.apache.log4j. PropertyConfigurator.doConfigure (String, LoggerRepository)[**void**]	7005	Parameter 2 of '**public void** doConfigure(java.lang.String, org. apache.log4j.Hierarchy)' has changed its type to org.apache.log4j.spi. LoggerRepository	**public**	N	1:n	Change Type	
org.apache.log4j. PropertyConfigurator.doConfigure (URL, Hierarchy)[**void**]	org.apache.log4j. PropertyConfigurator.doConfigure (URL, LoggerRepository)[**void**]	7005	Parameter 2 of '**public void** doConfigure(java.net.URL, org. apache.log4j.Hierarchy)' has changed its type to org.apache.log4j.spi. LoggerRepository	**public**	N	1:n	Change Type	
org.apache.log4j. PropertyConfigurator.doConfigure (Properties, Hierarchy)[**void**]	org.apache.log4j. PropertyConfigurator.doConfigure (Properties, LoggerRepository)[**void**]	7005	Parameter 2 of '**public void** doConfigure(java.util.Properties, org.apache.log4j.Hierarchy)' has changed its type to org.apache.log4j.spi. LoggerRepository	**public**	N	1:n	Change Type	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j. RollingFileAppender. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j. RollingFileAppender.setFile(String, **boolean**)[**void**]	org.apache.log4j. RollingFileAppender.setFile(String, **boolean**, **boolean**, **int**)[**void**]	7004	In method '**public void** setFile(java.lang.String, **boolean**)' the number of arguments has changed	**public**	N	1:n	Add Contextual Data	
org.apache.log4j. RollingFileAppender. setMaxFileSize(**long**)[**void**]	org.apache.log4j. RollingFileAppender. setMaximumFileSize(**long**)[**void**]	7002	Method '**public void** setMaxFileSize(**long**)' has been removed	**public**	Y	1:1	Rename	
org.apache.log4j. RollingFileAppender.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.RollingFile AppenderBeanInfo	org.apache.log4j. RollingFileAppender. setMaximumFileSize(**long**)[**void**]	8001	Class org.apache.log4j. RollingFileAppenderBeanInfo removed	**public**	N	n:1	Consolidate	In v1.1b3, this class got added as a workaround to an issue with one particular field on the RollingFileAppender class: setMaxFileSize wasn't compatible with the JAVABEANS paradigm because it was overloaded with **long** and String parameters: beans could only look up one method, not both. With the addition of setMaximumFileSize(**long**) to work around this, this was no longer needed. So basically, a lot of code was replaced with just a getter/setter.
org.apache.log4j.SimpleLayout. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.SimpleLayout. setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.WriterAppender.getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.WriterAppender.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang.String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.helpers.Loader.getResource(String, Class)[URL]	org.apache.log4j.helpers.Loader.getResource(String)[URL]	7004	In method '**public** java.net.URL getResource(java.lang.String, java.lang.Class)' the number of arguments has changed	**public**	N	1:1	Remove parameter	
org.apache.log4j.helpers.OnlyOnceErrorHandler.getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.helpers.OnlyOnceErrorHandler.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang.String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.helpers.OptionConverter.toPriority(String, Priority)[Priority]	org.apache.log4j.helpers.OptionConverter.toLevel(String, Level)[Level]	7002	Method '**public** org.apache.log4j.Priority toPriority(java.lang.String, org.apache.log4j.Priority)' has been removed	**public**	N	1:n	Rename; Change Type	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.helpers.OptionConverter.selectAndConfigure(URL, Hierarchy)[**void**]	org.apache.log4j.helpers.OptionConverter.selectAndConfigure(URL, String, LoggerRepository)[**void**]	7004	In method '**public void** selectAndConfigure(java.net.URL, org.apache.log4j.Hierarchy)' the number of arguments has changed	**public**	N	1:n	Change Type	
org.apache.log4j.helpers.SyslogQuietWriter.setPriority(**int**)[**void**]	org.apache.log4j.helpers.SyslogQuietWriter.setLevel(**int**)[**void**]	7002	Method '**public void** setPriority(**int**)' has been removed	**public**	N	1:1	Rename	
org.apache.log4j.net.JMSAppender.getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String[] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.net.JMSAppender.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang.String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.net.SMTPAppender.getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String[] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.net.SMTPAppender.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang.String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.net. SocketAppender. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.net. SocketAppender.setOption(String , String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.net.SocketNode. SocketNode(Socket, Hierarchy)	org.apache.log4j.net.SocketNode. SocketNode(Socket, LoggerRepository)	7005	Parameter 2 of '**public** SocketNode(java.net.Socket, org. apache.log4j.Hierarchy)' has changed its type to org.apache.log4j.spi. LoggerRepository	**public**	N	1:n	Change Type	
org.apache.log4j.net. SyslogAppender. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.net. SyslogAppender.setOption(String , String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.nt. NTEventLogAppender. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.nt.NTEventLogAppender.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang.String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
none	org.apache.log4j.spi.AppenderAttachable.isAttached(Appender)[**boolean**]	7012	Method '**public boolean** isAttached(org.apache.log4j.Appender)' has been added to an interface	**public**	N	1:n	Add method to interface	
org.apache.log4j.spi.CategoryFactory	org.apache.log4j.spi.LoggerFactory	8001	Class org.apache.log4j.spi.CategoryFactory removed	**public**	N	1:n	Change Type	
org.apache.log4j.spi.Configurator.doConfigure(URL, Hierarchy)[**void**]	org.apache.log4j.spi.Configurator.doConfigure(URL, LoggerRepository)[**void**]	7005	Parameter 2 of '**public void** doConfigure(java.net.URL, org.apache.log4j.Hierarchy)' has changed its type to org.apache.log4j.spi.LoggerRepository	**public**	N	1:n	Change Type	
none	org.apache.log4j.spi.ErrorHandler.error(String, Exception, **int**, LoggingEvent)[**void**]	7012	Method '**public void** error(java.lang.String, java.lang.Exception, **int**, org.apache.log4j.spi.LoggingEvent)' has been added to an interface	**public**	N	1:n	Add method to interface	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
none	org.apache.log4j.spi.ErrorHandler .setAppender(Appender)[**void**]	7012	Method '**public void** setAppender(org. apache.log4j.Appender)' has been added to an interface	**public**	N	1:n	Add method to interface	
none	org.apache.log4j.spi.ErrorHandler .setBackupAppender(Appender)[**void**]	7012	Method '**public void** setBackupAppender (org.apache.log4j.Appender)' has been added to an interface	**public**	N	1:n	Add method to interface	
none	org.apache.log4j.spi.ErrorHandler .setLogger(Logger)[**void**]	7012	Method '**public void** setLogger (org.apache.log4j.Logger)' has been added to an interface	**public**	N	1:n	Add method to interface	
org.apache.log4j.spi.Filter. getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.spi.Filter. setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang. String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.spi. LoggingEvent:category[Category]	org.apache.log4j.spi. LoggingEvent:logger[Category]	6001	Removed field category	**protected**	N	1:1	Rename	Not a typo; they changed the name, but I think they forgot to also change the field's type.

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.spi.LoggingEvent:priority[Priority]	org.apache.log4j.spi.LoggingEvent:level[Priority]	6001	Removed field priority	**protected**	N	1:1	Rename	Not a typo; they changed the name, but I think they forgot to also change the field's type.
org.apache.log4j.spi.OptionHandler.getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.spi.OptionHandler.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang.String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.spi.RootCategory.RootCategory(Priority)	org.apache.log4j.spi.RootCategory.RootCategory(Level)	7005	Parameter 1 of '**public** RootCategory(org.apache.log4j.Priority)' has changed its type to org.apache.log4j.Level	**public**	N	1:n	Change Type	
org.apache.log4j.spi.RootCategory.setPriority(Priority)[**void**]	org.apache.log4j.spi.RootCategory.setPriority(Level)[**void**]	7005	Parameter 1 of '**public void** setPriority(org.apache.log4j.Priority)' has changed its type to org.apache.log4j.Level	**public**	N	1:n	Change Type	
org.apache.log4j.varia.ExternallyRolledFileAppender.getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.varia.ExternallyRolledFileAppender.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang.String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.varia.PriorityMatchFilter	org.apache.log4j.varia.LevelMatchFilter	8001	Class org.apache.log4j.varia.PriorityMatchFilter removed	**public**	N	1:n	Change Type	
org.apache.log4j.varia.PriorityRangeFilter	org.apache.log4j.AppenderSkeleton.setThreshold(Priority)[**void**]	8001	Class org.apache.log4j.varia.PriorityRangeFilter removed	**public**	N	n:1	Consolidate	The replacement is confusing, until you look at the architecture. The old BI is a Filter, and Filters are submitted to Appenders, where the Appenders use them to do the filtering. So instead of submitting this Filter, you set the threshold on the Appender to what you want. Note that this eliminates the Max threshold, probably because the devs didn't think you should be able to ignore FATAL events. =D
org.apache.log4j.xml.DOMConfigurator	org.apache.log4j.xml.DOMConfigurator	5001	Removed org.apache.log4j.BasicConfigurator from the list of superclasses	**public**	N	1:n	Supertype Change	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.xml.DOMConfigurator.parse(Element, Hierarchy)[**void**]	org.apache.log4j.xml.DOMConfigurator.parse(Element)[**void**]	7004	In method '**protected void** parse(org.w3c.dom.Element, org.apache.log4j.Hierarchy)' the number of arguments has changed	**protected**	N	1:1	Remove parameter	
org.apache.log4j.xml.DOMConfigurator.parseCategory(Element, Hierarchy)[**void**]	org.apache.log4j.xml.DOMConfigurator.parseCategory(Element)[**void**]	7004	In method '**protected void** parseCategory(org.w3c.dom.Element, org.apache.log4j.Hierarchy)' the number of arguments has changed	**protected**	N	1:1	Remove parameter	
org.apache.log4j.xml.DOMConfigurator.parseChildrenOfCategoryElement(Element, Category, **boolean**)[**void**]	org.apache.log4j.xml.DOMConfigurator.parseChildrenOfCategoryElement(Element, Logger, **boolean**)[**void**]	7002	Method '**protected void** parseChildrenOfCategoryElement(org.w3c.dom.Element, org.apache.log4j.Category, **boolean**)' has been removed	**protected**	N	1:n	Change Type	
org.apache.log4j.xml.DOMConfigurator.parsePriority(Element, Category, **boolean**)[**void**]	org.apache.log4j.xml.DOMConfigurator.parsePriority(Element, Logger, **boolean**)[**void**]	7002	Method '**protected void** parsePriority(org.w3c.dom.Element, org.apache.log4j.Category, **boolean**)' has been removed	**protected**	N	1:n	Change Type	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.xml.DOMConfigurator.parseRenderer(Element, Hierarchy)[**void**]	org.apache.log4j.xml.DOMConfigurator.parseRenderer(Element)[**void**]	7004	In method '**protected void** parseRenderer(org.w3c.dom.Element, org.apache.log4j.Hierarchy)' the number of arguments has changed	**protected**	N	1:1	Remove parameter	
org.apache.log4j.xml.DOMConfigurator.parseRoot(Element, Hierarchy)[**void**]	org.apache.log4j.xml.DOMConfigurator.parseRoot(Element)[**void**]	7004	In method '**protected void** parseRoot(org.w3c.dom.Element, org.apache.log4j.Hierarchy)' the number of arguments has changed	**protected**	N	1:1	Remove parameter	
org.apache.log4j.xml.DOMConfigurator.doConfigure(InputStream, Hierarchy)[**void**]	org.apache.log4j.xml.DOMConfigurator.doConfigure(InputStream, LoggerRepository)[**void**]	7005	Parameter 2 of '**public void** doConfigure(java.io.InputStream, org.apache.log4j.Hierarchy)' has changed its type to org.apache.log4j.spi.LoggerRepository	**public**	N	1:n	Change Type	
org.apache.log4j.xml.DOMConfigurator.doConfigure(String, Hierarchy)[**void**]	org.apache.log4j.xml.DOMConfigurator.doConfigure(String, LoggerRepository)[**void**]	7005	Parameter 2 of '**public void** doConfigure(java.lang.String, org.apache.log4j.Hierarchy)' has changed its type to org.apache.log4j.spi.LoggerRepository	**public**	N	1:n	Change Type	

Continued on next page

Table E.12 — continued from previous page

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.xml.DOMConfigurator.doConfigure(URL, Hierarchy)[**void**]	org.apache.log4j.xml.DOMConfigurator.doConfigure(URL, LoggerRepository)[**void**]	7005	Parameter 2 of '**public void** doConfigure(java.net.URL, org.apache.log4j.Hierarchy)' has changed its type to org.apache.log4j.spi.LoggerRepository	**public**	N	1:n	Change Type	
org.apache.log4j.xml.XMLLayout.getOptionStrings()[String[]]	none	7002	Method '**public** java.lang.String [] getOptionStrings()' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.
org.apache.log4j.xml.XMLLayout.setOption(String, String)[**void**]	none	7002	Method '**public void** setOption(java.lang.String, java.lang.String)' has been removed	**public**	Y	1:n	Extract to External API	Release notes say these are redundant with the move to JAVABEANS configuration.

Table E.13: LOG4J 1.2.1 to 1.2.2 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.helpers.Loader. loadClass(Double)[Class]	org.apache.log4j.helpers.Loader. loadClass(String)[Class]	7005	Parameter 1 of ’**public** java.lang.Class loadClass(java.lang.Double)’ has changed its type to java.lang.String	**public**	N	1:n	Change Type; Repair Damaged Code	In Revision 309416 of the v1.2 release, this method was deliberately damaged by a developer to avoid its use. All code inside was commented out, the parameter type was changed, and the method returns **null**. This is the fix. The issue appears to have a bug identified, and then the fix was still in debate when this release went out—the developer damaged the code to prevent use while he reasoned about a fix, and then either didn’t realize it was damaged when released, or wanted it that way. The relevant bug report is https://issues.apache.org/bugzilla/show_bug.cgi?id=9305.

Table E.14: LOG4J 1.2.4 to 1.2.5 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.lf5.util.ProductProperties	none	8001	Class org.apache.log4j.lf5.util.ProductProperties removed	**public**	N	Delete	Delete	Deleted in response to to the removal of the Help:About dialog from the SWING GUI.
org.apache.log4j.lf5.viewer.LogBrokerMonitor.createHelpAbout()[JMenuItem]	none	7002	Method '**protected** javax.swing.JMenuItem createHelpAbout()' has been removed	**public**	N	Delete	Delete	Removed this dialog which just showed version details about the SWING GUI for LOG4J.
org.apache.log4j.lf5.viewer.LogFactor5AboutDialog	none	8001	Class org.apache.log4j.lf5.viewer.LogFactor5AboutDialog removed	**public**	N	Delete	Delete	Deleted in response to to the removal of the Help:About dialog from the SWING GUI.

Table E.15: LOG4J 1.2.5 to 1.2.6 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.net.JMSSink. JMSSink()[JMSSink]	org.apache.log4j.net.JMSSink. JMSSink(String, String, String, String)[JMSSink]	7004	In method '**public** JMSSink()' the number of arguments has changed	**public**	N	1:n	Add contextual data; Encapsulate	A constructor was added that took four string parameters, as part of a class rewrite. Since there was no constructor previously, the implicit default constructor was there, and so this is effectively a change to the default constructor.
org.apache.log4j.spi. LoggingEvent:logger[Category]	org.apache.log4j.spi. LoggingEvent.getLoggerName()[String]; org.apache.log4j.Category. getInstance(String)[Category]	6010	Accessibility of field logger has been weakened from **public** to **private**	**public**	N	1:n	Hide; Encapsulate (Partial)	LOG4J was transitioning away from Category to the Level type, so it hid this field. To get a reference to it, you need to get the category name, and then call up an instance of it. Note that this uses deprecated methods; the proper replacement is through log4j.Logger.getLogger().

Table E.16: LOG4J 1.2.8 to 1.2.9 migration.

Original	Replacement(s)	BI Code	BI Description	Vis	Depr?	Classif	Xform	Notes
org.apache.log4j.config.PropertyPrinter.printOptions(PrintWriter, Category)[**void**]	org.apache.log4j.config.PropertyPrinter.printOptions(PrintWriter, Logger)[**void**]	7005	Parameter 2 of '**protected void** printOptions(java.io.PrintWriter, org.apache.log4j.Category)' has changed its type to org.apache.log4j.Logger	**protected**	N	1:n	Change Type	Since it's being downcast, isn't just a straight substitution.