UNIVERSITY OF CALGARY

Using Method Similarity over Versions to Improve Predictions Based on Change History

by

Bhavya Rawal

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

August, 2009

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 "Using Method Similarity over Versions to Improve Predictions Based on Change History" submitted by Bhavya Rawal in partial fulfillment of the requirements for the degree of Master of Science.

Dr. Robert James Walker
Supervisor, Department of Computer Science

Dr. Jörg Denzinger
Department of Computer Science

Dr. Behrouz Homayoun Far
Department of Electrical & Computer Engineering

Date

Abstract

Software developers are often required to perform modification tasks that involve source code spanning multiple files. To assist developers in finding relevant source code when working on a modification task, current approaches make use of data mining techniques to identify change patterns. These change patterns are mined from the version history of a system and are used to identify co-changing entities of potential interest.

These current approaches rely on the name and location of the source code entity to identify that entity and to associate history with it. Using name and location for entity identification leads to loss of historical information for entities that have undergone any transformations that have modified their name or location. Such transformations occur during common maintenance practice (like refactoring) and are fairly commonplace in evolving systems.

In our work we have focused on associating history with transformed code entities to improve mining-based recommender systems. We have used various method facts to identify similarity between methods in different versions. Historical information from highly similar entities is used to estimate the historical information that is missing from transformed entities. We hypothesize that predictions utilizing this similarity-derived historical information can provide higher number of pertinent recommendations over techniques that would not estimate the missing information.

To test our hypothesis, we have extended the existing change history based approach of Ying et al. and compared the results of our extension with the original approach.

Table of Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Software systems are subject to constant change [9, 32]. Post-development, these changes fall under four categories of "maintenance": *corrective*, *adaptive*, *perfective*, and *preventive* [33]. Corrective maintenance involves modifications made to the system to fix errors detected in the system. Adaptive maintenance involves changes made to the system to make it compatible with its external environment, e.g., there may be changes to the host machine's operating system resulting in changes to the software running on the machine. Perfective maintenance is performed to improve the functionality of the system. Preventive maintenance is performed to ease the future maintenance of the system.

In terms of actual implementation these activities manifest themselves in the form of modification tasks such as bug fixes and feature implementations. For non-trivial systems, these tasks require considerable planning, effort, and time. Successful performance of these tasks entails correct determination of the scope of the task, which further involves finding a subset of code that will be changed as part of the task. In practice, a modification task requires changing parts of code which the developer knows would be affected and various secondary changes which are due to the "ripple effect" of the first set of changes.

Typically, a version control system (VCS)—like Concurrent Versions System (CVS)[1], Subversion (SVN)[2], or Microsoft Visual SourceSafe[3]—is used to manage the co-development of evolving software systems, by implementing a protocol that is invoked when multiple developers make changes to the same piece of code. The protocol tries to merge the changes and calls attention to any conflicts, thereby ensuring that there is always a central, conflict-free copy of

[1] http://www.nongnu.org/cvs
[2] http://subversion.tigris.org/
[3] http://en.wikipedia.org/wiki/Visual_SourceSafe

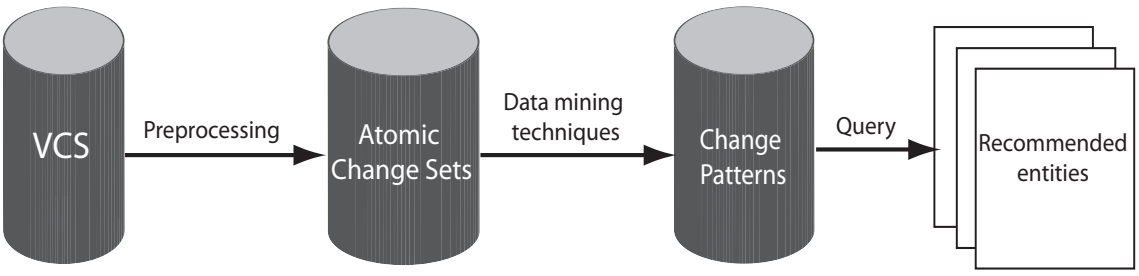

Figure 1.1: Stages in change history based recommender systems.

the code maintained.

One of the important side effects of an ever changing system managed using a VCS is a rich history of changes of the system. A system managed using a VCS has all its changes stored in a central repository. One way to leverage this rich, detailed history of a software system is as guidance to developers working on modification tasks.

Currently, various approaches are available that help developers in determining the subset of code which would likely have to be changed as part of a modification task. Some of these existing approaches [41, 44, 38] help developers by suggesting parts of code that are likely to change given an initial set of changes for a modification task. These approaches leverage the code history or current structure of the system or both. Together they form the category of change history based (CHB) recommender systems.

Figure 1.1 describes the workings of a typical CHB recommender system. In the first stage, changes stored in the VCS as files and line numbers are extracted in the form of atomic change sets. A change set is a *set of source code entities that are changed together*. In SVN there are explicit change sets that are generated on each commit whereas in CVS the change sets must be inferred; this is often performed by extracting the source code entities changed within a 3-minute time interval by the same author and annotated with the same comment [35]. The timeframe and other heuristics (author and comment similarity) are used to attempt to capture the entities that were changed together as part of a common development goal.

In the next stage, various data mining techniques are applied to the extracted change sets

to obtain interesting patterns and associations between source code entities. An example of the patterns obtained by these techniques is: If `Authentication.java` is changed then `UserPrivileges.java` is changed too; or more explicitly, `Authentication.java` and `UserPrivileges.java` changed 45 times together when `Authentication.java` changed 50 times and `UserPrivileges.java` changed 55 times. Alternatively the pattern can be at a finer granularity too. For example, the `init()` method in `Authentication.java` changed with the `init()` method in `UserPrivileges.java`.

Next the extracted change patterns are used to obtain recommendations for developers working on change tasks. This is done by querying the set of change patterns to obtain patterns of interest for the modification task at hand. Thus if the modification task involves modifying `Authentication.java` (continuing the above example) the developer will be pointed to `UserPrivileges.java`, as both files are involved in a pattern.

One of the biggest advantages of existing CHB recommender techniques is that change patterns can capture some of the logical associations among source code entities that are not structurally related. Hence, while regular static and dynamic analyses [4, 39] might fail to detect the association between non-structurally related entities, change history based techniques can find useful non-structural associations based on the usage of these entities. And sometimes, change patterns are able to capture logical associations between files written in different languages. In addition, change patterns can help developers pare down a large number of relevant entities by identifying stronger associations between a select few. Thus, the recommender systems help developers working on modification tasks.

In many existing CHB recommender systems [41, 44, 38], source code entities are identified uniquely based on their name and location in the system. This is specifically a limitation of CVS, as CVS does not support tracking of renames or moves, resulting in the moved/renamed entity's history being split into two. For our work we have focused on systems maintained using CVS.

One issue with the current approaches is that they do not take into account source code entity (classes, methods, etc. in Java; functions in C++) transformations resulting from software maintenance activities like refactoring [41, 44]. Transformations can result in changed name or location or both for a particular entity. The transformed entity is now classified as a new entity, which results in loss of history.

Figure 1.2 depicts three such transformations from version $n-1$ to version n of a system.

- Entity `A` in version $n-1$ is split into entities `A1` and `A2` in version n.
- Entities `B` and `C` in version $n-1$ are combined into entity `BC` in version n.
- Entity `D` in version $n-1$ is renamed to entity `newD` in version n.

Currently, there are various approaches that track the evolution of a software system over time by detecting changes at the level of individual source code entities. These approaches track code clones to detect the evolution of similar code [27] or use facts corresponding to software entities to detect the origin of an entity [45] or apply pre-defined rules to detect transformations [26]. While these approaches are helpful in detecting changes to a given entity over a period of time and are thus helpful in understanding the evolution of the system as a whole, none of these approaches are used by the change history mining systems to associate history with transformed entities.

[BR: Model-based approaches use models ~~generated from~~ software artifacts to guide developers in a software development exercise. For example, model-based testing approaches provide ways for automatic test case generation based on software models [16]. In theory model-based techniques can be applied to recommender approaches. Models used for automatic test generations ~~can~~ be modified to provide recommendations to developers working on code modification. UML diagrams ~~can~~ serve as a model of a system representing all the dependencies. However, software artifacts like UML diagrams and

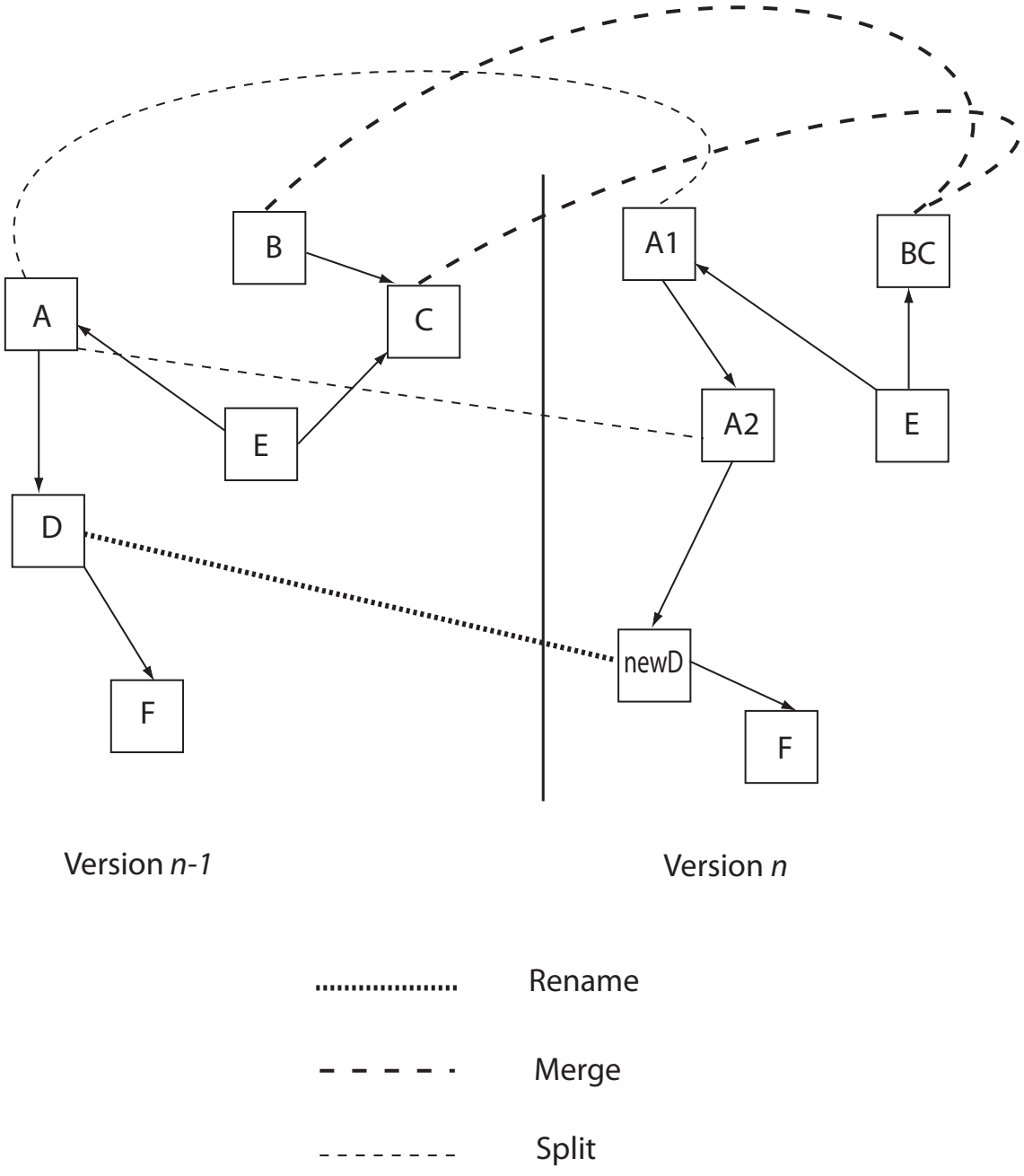

Figure 1.2: Different source code transformations from version $n-1$ to version n.

documentation are not updates at regular intervals resulting out-of-date and often incorrect documentation [31, 11, 37]. Thus, for most systems model-based recommender approaches ~~are not practical~~.]

In order to associate useful history with transformed entities we have developed a transformation detection extension to an existing change history based approach by Ying et al. [41]. Ying et al.'s approach uses frequent pattern mining [22], "a data-mining technique for obtaining patterns with high frequency" to identify change patterns above a given threshold of frequency. While Ying et al.'s approach showed promise in recommending relevant source code entities for change tasks, the precision (measure of correctness) and recall (measure of completeness) rates for the two benchmark systems used to evaluate the approach (Eclipse[4] and Mozilla[5]) were low (precision = 30% recall = 10–20% and precision = 50% recall = 20–30% respectively). We describe reasons for selecting Ying et al.'s approach in the Discussion chapter of this thesis. In principle our extended approach should provide comparable results for any change history based approach.

One of the issues with the current change history based recommender systems is that they do not take into account activities like refactoring, which result in loss of history for the refactored entities. Ying et al. acknowledge this issue in their journal publication [41, p. 583]:

> Moreover, significant rewrites and refactoring of the code base can affect our approach. Such changes affect the patterns we compute because we do not track the similarity of code across such changes.

Another concern raised by Ying et al. addresses their use of files as the unit of change patterns as opposed to finer entities like methods [41, p. 583]:

> Applying our approach to methods where change patterns describe methods instead of files that change together repeatedly may provide better results because a

[4]http://www.eclipse.org
[5]http://www.mozilla.org

smaller unit of source code may suggest a similar intention behind the separated code.

To investigate the importance of detecting transformation and finer granularity in CHB recommender systems, we have extended Ying et al.'s original approach. Specifically, we have developed an algorithm to detect method based similarity over various versions of a software system. This similarity information is then combined with change patterns to derive predictions. Next, we extend the original file based approach to include transformation information. The two extended approaches (file based and method based) allow us to investigate the role of both transformations and granularity in CHB recommender systems. We thus investigate the effectiveness of three approaches in this thesis for two granularities resulting in 6 approaches:

1. Ying et al.'s CHB approach for file-level granularity,
2. Ying et al.'s CHB approach for method-level granularity,
3. similarity based (SB) approach for class-level granularity,
4. SB approach for method-level granularity,
5. extended change history based (ECHB) approach (including transformation information) for class-level granularity, and
6. ECHB approach (including transformation information) for method-level granularity.

We often use the terms "file-level" and "class-level" interchangeably as we focus on Java source code and, generally, each Java class resides in its own file. Furthermore, CHB approaches are not aware of syntactic structures (like classes) whereas our 2 SB approaches utilize the syntactic structures and are more appropriately termed "class-level".

To evaluate the 6 approaches, we perform a series of experiments using the Eclipse project as our benchmark system. Our experiments use real modification tasks, used by Ying et al. in

their original experiments, from Eclipse code base. The experiments were designed to compare results for the following categories:

1. Ying et al.'s CHB recommender approach versus our SB recommender approach and ECHB recommender approach.

2. Effect of granularity (method level and file level) on the quality of recommendations.

1.1 Thesis Statement

The thesis of this work is that by using method similarity over versions to detect transformations and using this information to build an extension to Ying et al.'s CHB recommender approach, we can fill historical gaps in their approach and provide valid recommendations when an entity is transformed, thereby expanding the scope of their recommender approach.

1.2 Thesis Organization

The thesis begins with Chapter 2, describing our partial replication of Ying et al.'s original file level CHB recommender approach. Chapter 3 describes our implementation of Ying et al.'s CHB approach for method-level granularity. Chapter 4 describes our algorithm for detecting method-level similarity over two versions and its implementation as a SB recommender approach. Chapter 5 describes the class-level implementation of our SB approach. Chapter 6 describes our ECHB approach, for method level and class level granularity, leveraging the similarity algorithm to fill in gaps for missing change history. Chapter 7 summarizes issues around our implementation of the various approaches and their evaluation. Chapter 8 compares our work with other work in the field. Additional points and future work are discussed in Chapter 9.

Chapter 2

PARTIAL REPLICATION OF YING ET AL.'s APPROACH

In this chapter, we describe our re-implementation and re-validation of Ying et al.'s change history based recommender approach [41]. As Ying et al.'s approach is not available as a tool, in order to extend it, we needed to replicate a large portion of their work. Our replication is based on four sources of information: their journal publication [41], Ying's MSc thesis [42], an email correspondence with Ying, and a teleconference meeting with Ying.

In this chapter, we first describe their original approach in Section 2.1. In Section 2.2 we describe our re-implementation. In Section 2.3 we present the findings from our attempts at replicating a subset of their original evaluation, including several variations that were tried where particular points were ambiguous.

2.1 Stages in the Approach

Ying et al.'s approach works at the file level, and consists of three stages:

1. Data is extracted from a version control system like CVS and preprocessed to eliminate spurious changes and to organize individual file level changes into *atomic change sets*.

2. The atomic change sets obtained from the first stage are used as an input for a data mining algorithm to extract frequent patterns.

3. The extracted frequent patterns are queried to recommend relevant source files to developers working on a modification task.

We examine these stages in turn.

2.1.1 Stage 1: Data preprocessing

The data mining algorithm used by Ying et al. takes as input a set of atomic change sets to return all frequent patterns. An atomic change set is a set of files that were committed to a version control repository in a single transaction. Unfortunately, some version control systems (such as CVS, which is heavily used in industrial practice) do not explicitly record atomic change sets, but rather record only the individual file changes. The atomic change sets must be estimated from other evidence present in the repository.

To obtain change history in the form of atomic change sets, we need heuristics to map file-level changes to atomic change sets. Ying et al. create atomic change sets based on the temporal proximity of file changes and certain metadata associated with the change: two or more files that changed within a three-minute interval by the same author and checked in using the same comment are considered to be part of an atomic change set. This technique was first described by Mockus et al. [35].

Next, all change sets of cardinality 100 or more are removed from the extracted change sets to eliminate any transaction that likely were the result of large-scale activities (such as merging an entire branch of the repository into the main trunk) and not representative of regular developmental changes. These change sets are now ready to be mined for interesting patterns.

2.1.2 Stage 2: Frequent pattern mining

Ying et al. use the FP-tree mining algorithm [22], a form of association rule mining, to extract frequent patterns from the change history of a system.

Association rule mining is a data mining technique used to detect relationships between items in a database of transactions [5]. The technique was first used on supermarket transactional databases to identify buying patterns of customers, which were then used for specific product placements, e.g., *90% of customers who buy beer also buy diapers* (resulting in beer and diapers being placed in closer aisles). Ying et al. have used frequent pattern mining to find

source code files that frequently change together.

Frequent pattern mining, in Ying et al.'s approach, identifies files that change together a sufficient number of times, where the measure of sufficiency is a user-provided threshold value. A pattern is of the form $P = \{f_1, f_2, \ldots, f_n\}$ and the number of times the pattern was repeated in a collection of change sets is called its *support* s. The user-provided threshold is the minimum support for which the pattern is considered frequent. We refer to this threshold as frequent threshold.

Frequent pattern mining consists of two stages [22]: construction of an FP-tree and mining the FP-tree for frequent patterns. The FP-tree construction algorithm takes as input a list of atomic change sets and the user-provided minimum support threshold, and returns an FP-tree data structure as a compact representation of the atomic change sets. Next, the FP-tree growth algorithm takes as input the previously constructed FP-tree and the user-provided minimum support threshold and returns a *complete set of frequent patterns*.

In this section we highlight certain features of the FP-tree construction and FP-tree growth, together referred to as FP-tree mining, to describe the modifications to the algorithm by Ying. For a detailed description of the original FP-tree mining algorithm, please refer to Appendix A in Ying's MSc thesis [42, p. 56] or the original work by Han et al. [22]; note that Ying's modifications to the algorithm are not described in Ying et al.'s journal publication. The details of Ying et al.'s modifications are provided in Ying's MSc thesis [42, p. 13], and are as follows.

> We have modified the FP-tree mining algorithm so that we do not continue to generate subsets of patterns from a single-branched tree. In addition, we only output maximal patterns with respect to D from the mining process, as opposed to all patterns and their subsets. Storing only the maximal patterns does not affect the recommendations generated by the patterns because recommendations generated by all patterns are contained in the recommendations generated using the maximal patterns.

Note that a pattern in D is *maximal* if it is not a subset of any other pattern in D. We refer to Ying's description of these modifications, as pertinent, in our brief description of FP-tree mining.

Tree construction. The FP-tree mining algorithm processes a collection of atomic change sets to extract frequent patterns in the collection. For the sake of brevity of explanation, assume that the files in each change set in the collection are ordered in descending order of the frequency count of the files in the collection. In addition, assume that the files whose frequency count is less than the user-provided minimum support threshold are eliminated from all the change sets. The FP-tree construction algorithm does not make these assumptions but in effect, transforms the change sets to this form during tree creation. At the end of the FP-tree construction algorithm, the collection of atomic change sets is represented by the FP-tree.

For some parts of the algorithm we provide a running example based on a similar one provided by Han et al. [22], for which Table 2.1 gives a collection of atomic change sets. The first column gives the original change sets and the second column gives sorted and modified change sets.

Change sets	Sorted and modified change sets
g, b, d, e, h, j, n, q	g, d, b, n, q
b, c, d, g, m, n, p	g, d, b, c, n
c, g, i, k, p	g, c
c, d, l, t, q	d, c, q
b, g, d, f, m, q, n, o	g, d, b, n, q

Table 2.1: Running example's change set (frequent threshold = 3).

The general organization of the FP-tree is as follows. The root node of the FP-tree is labelled *null*, all other nodes in the tree represent the files in the change sets and are labelled with a *count* value. (Note that *count* is not the frequency count of the file in the collection of change sets.) The tree has the following properties:

- Files in the change sets may be placed in one or more nodes in the tree. Also, for a given file, the sum of the counts in these nodes equals the frequency count of the file in the collection of change sets.

- Every change set $cs = (a_1, a_2, ..., a_k)$ in the collection corresponds to a path in the tree (root node $\rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_k$)

- If the first l files in the two change sets cs_1 and cs_2 are the same, then the paths that correspond to patterns cs_1 and cs_2 share the first l (excluding the root) nodes in the tree.

- Let (root node $\rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_k$) be the path to node a_k in the FP-tree. Then, the *count* of node a_k is the number of change sets for which the first k files, in order, are a_1, a_2, ..., a_k.

- If the first file in two patterns cs_1 and cs_2 are not the same, then they share only the root node.

- In addition, the FP-tree contains a *header table* that stores the individual frequent files in the decreasing order of their frequency in the first column. The second column links each file in the header table to all its corresponding occurrences in the tree.

Figure 2.1(a) gives the FP-tree generated by the FP-tree construction algorithm for the change sets in Table 2.1.

Frequent pattern mining from the FP-tree. The general principle of FP-growth is divide-and-conquer, as follows.

A *suffix pattern* is a set of files, used as a starting point to build an FP-tree; FP-growth starts with suffix patterns of length 1. In Ying et al.'s approach suffix patterns of length 1 are individual files, selected one at a time from the header table. For each suffix pattern, its *conditional pattern base* or all the nodes in the tree from the root node up to but not including

the suffix pattern, is obtained. For example, the greyed nodes in Figure 2.1(a) illustrates the conditional pattern base for suffix pattern n. This conditional pattern base consists of the two paths $(g \rightarrow d \rightarrow b)$ with *count* of 2, and $(g \rightarrow d \rightarrow b \rightarrow c)$ with *count* of 1.

Next, a smaller FP-tree, called the *conditional FP-tree*, is created from the conditional pattern base by invoking the FP-tree construction algorithm on the conditional pattern base. This step is applied recursively until the resulting smaller FP-tree consists of only a single path. All combinations of files in this single path tree are returned as frequent patterns. Initial suffix patterns grow by appending them with the newer ones generated from the conditional FP-tree. Figure 2.1(b) illustrates the result.

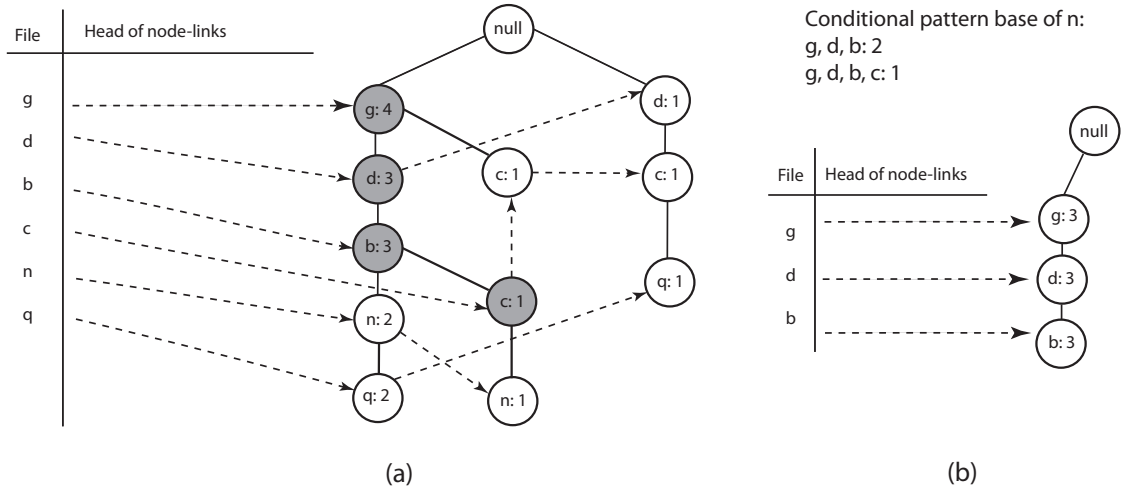

Figure 2.1: FP-tree construction and growth, showing (a) the conditional pattern base for suffix pattern n, and (b) the conditional FP-tree created from this conditional pattern base.

Ying modified the algorithm to stop the generation of frequent patterns from a single branched tree (a tree with a single path between its root node and leaf node), and to generate only maximal frequent patterns [42, p. 13]. From Ying's thesis, the modification needed to achieve the first of the two changes described above is clear but the modification needed to achieve the second of the two changes is not. We have made this first change in our implemen-

tation of the algorithm.

Note that the first modification to the FP-tree mining algorithm is not sufficient to produce maximal patterns on its own, as the FP-growth algorithm produces frequent patterns at two steps. First, when patterns are generated from a single path tree and second, when suffix patterns are output as frequent patterns. Both these steps produce frequent patterns not all of which are maximal. Thus, in our understanding of the FP-growth algorithm, it would need to be changed non-trivially and for several steps to generate only maximal patterns.[1] Henceforth in this thesis we use the phrase "*our modified FP-tree mining algorithm*" to indicate the use of FP-tree mining algorithm with a single modification and "*Ying et al.'s modified FP-tree mining algorithm*" to indicate Ying et al.'s two modifications.

Indeed, the problem of mining maximal patterns is well-known and well-researched in the field of data mining, but such work utilizes alternative algorithms [12, 19, 20]. Regardless, Ying et al. state that using only maximal patterns, in place of union of *all* patterns, would not change the recommendations [42, p. 13]. We achieved the same result by using only unique recommendations when evaluating individual modification tasks later in the experiment. Thus, the results from our modified FP-tree mining algorithm should be the same as those of Ying et al.'s modified FP-tree mining algorithm. The only difference would be that the computational complexity might be greater for our implementation. We discuss an alternative approach to obtain frequent patterns, which we used to corroborate the frequent patterns generated by our implementation of FP-tree mining, in Section 2.3.

2.1.3 Stage 3: Query

In this stage a developer working on a modification task queries the set of frequent patterns obtained in the previous stage for recommendations. The query consists of at least one file known to be involved in the solution of a particular modification task. The supplied input

[1] Even through our personal correspondence with Ying, we could not determine the exact change, made by Ying et al., to the algorithm.

files are then matched against the extracted frequent patterns to obtain patterns involving the input files. All the files thus identified (other than the input files) form a recommendation. Thus if $\{x, y\}$ is the input file set, matched against the frequent patterns $\{x, a, b\}$, $\{x, b, t, q\}$, and $\{y, c\}$, then $\{a, b, t, q, c\}$ would form the recommendation set. We define the function $\mathrm{Rec} : 2^F \rightarrow 2^F$ to take a set of input files and to return the set of recommended files (plus the set of input files, for notational convenience).

2.2 Experiment

Ying et al. evaluated the usefulness of their approach by using it to make predictions for real modification tasks in Eclipse[2] and Mozilla[3]. To test the correctness of our implementation of their approach, we re-created their experiments for Eclipse. We did not repeat their experiments for Mozilla, as our intended extension of their approach would work only for Java systems at this point, and Mozilla is implemented in C/C++.

In this section we describe the implementation of their approach and recreation of the experiments in detail. Any modifications that we made to their original approach along with their implications are discussed. We also discuss our interpretation of some of the less clear details.

2.2.1 Setup and measurements

To evaluate the usefulness of their approach, Ying et al. used 40 modification tasks randomly selected from Eclipse and Mozilla (20 tasks each) and applied their approach on these to recommend relevant files as solutions for the respective tasks. Their validation strategy involved comparing the code entities (files) changed as part of a known solution for a modification task against the recommended code provided by their approach. The set of entities changed as part of the solution to a modification task are referred to as the task's solution set. The usefulness

[2] www.eclipse.org
[3] www.mozilla.org

of the recommendations is then evaluated using the quantitative measures, *precision* and *recall* (they also used a subjective and qualitative measure called *interestingness* that we will not examine).

Informally, the *precision* and *recall* of a set of recommendations refer respectively to the exactness and the completeness of the results. They are calculated using the recommended set F_r and the known solution set F_s relative to a particular modification task m. Equations 2.1 and 2.2 define these measures more precisely.

$$\mathrm{pr}_m(F_r, F_s) = \frac{|F_s \cap F_r|}{|F_r|} \tag{2.1}$$

$$\mathrm{rc}_m(F_r, F_s) = \frac{|F_s \cap F_r|}{|F_s|} \tag{2.2}$$

For inputs that do not result in any recommendations we assign the following values: $\mathrm{pr}_m(\{\}, F_s) = 0$ and $\mathrm{rc}_m(\{\}, F_s) = 0$. Ying et al. do not state specifically how they evaluate $\mathrm{pr}_m(\{\}, F_s)$ and $\mathrm{rc}_m(\{\}, F_s)$. We discuss implications of assigning 0 in these situations and an alternate, better measure for representing these cases in Subsection 2.3.3.

Ying et al. combine the precision and recall values per input to obtain the average precision and recall [41, p. 577]:

> In the computation of the average measures, [...], we included only the precision and recall for recommendations from modification tasks M in the test data where each task's solution contained at least one file from a change pattern. We used each file f_s in the solution of a modification task $m \in M$ to generate a set of recommended files [F_r] and calculated [F_r's] precision and recall in the equations in [2.1 and 2.2]. The average precision is the mean of such precision values and analogously for the average recall.

We interpret this description such that, for each task m, we compute the mean precision

and recall values over all individual inputs, as shown in Equations 2.3 and 2.4 respectively.

$$\widetilde{\mathrm{pr}_m} = \frac{1}{|F_s|} \sum_{f_s \in F_s} \mathrm{pr}_m(\{f_s\}, F_s) \tag{2.3}$$

$$\widetilde{\mathrm{rc}_m} = \frac{1}{|F_s|} \sum_{f_s \in F_s} \mathrm{rc}_m(\{f_s\}, F_s) \tag{2.4}$$

Next, to compute $\mathrm{pr}_{\mathrm{avg}}$ and $\mathrm{rc}_{\mathrm{avg}}$, which is the final mean of all results, we compute the mean of $\widetilde{\mathrm{pr}_m}$ and $\widetilde{\mathrm{rc}_m}$ respectively, over all tasks, as shown in Equations 2.5 and 2.6. In Section 2.3.2 we discuss an alternative interpretation of Ying et al.'s description.

$$\mathrm{pr}_{\mathrm{avg}} = \frac{1}{|M|} \sum_{m \in M} \widetilde{\mathrm{pr}_m} \tag{2.5}$$

$$\mathrm{rc}_{\mathrm{avg}} = \frac{1}{|M|} \sum_{m \in M} \widetilde{\mathrm{rc}_m} \tag{2.6}$$

For their experiments, Ying et al. divide Eclipse's change history into training data and test data. Their general approach involves making recommendations using frequent patterns extracted from training data for the modification tasks selected from the test data. They select 20 modification tasks from Eclipse, reported and fixed, in the time period of the test data. The files fixed as part of the solution for the modification tasks are not recorded directly and hence Ying et al. use two common developer practices as heuristics to obtain the solution set for the tasks:

- The identifier for the modification task is used in the comments when checking in files involved in the solution. (Note that a given modification task involved performing a change request or repairing an error, as described by a "bug report" in Bugzilla, identified with a unique number; thus we often refer to bugs and their associated modification tasks interchangeably.)

- Developers commit the files associated with a modification task in the repository around the same time when the status of the modification task is changed to "fixed" in Bugzilla. (We discuss the shortcomings of this heuristic in Section 2.2.2.)

After repeating the experiments for thresholds 5, 10, 15, and 20, they report that a frequent threshold value of 10 provided the best tradeoff in terms of precision and recall values for frequent pattern mining from the training data.

2.2.2 Partial replication

We extracted the history for all Eclipse projects of the form `org.eclipse.*`. Apart from these, the Eclipse code base also contains external projects like `org.apache.ant` and `org.apache.lucene`, which we ignored. Ying confirmed in our personal correspondence that this corresponds to the code base that they analyzed.

Next, we extracted atomic change sets based on the training data timeline provided by Ying et al. This training period was from 2001/03/28 (initial check-in) to 2002/10/01, and the test period was from 2002/10/01 to 2003/05/01. To ensure that the projects are extracted today as they existed then, we created a CVS *date tag* for 2002/10/01 (the end of the training data) and extracted history relative to this date tag. Thus, for all the files existing on 2002/10/01, history was extracted. This history consisted of all the changes from the initial creation of the file until the date of extraction. We then truncated the extracted change sets to include only history until 2002/10/01. In our initial extractions, we experienced a timeout problem with the Eclipse server for about fifteen files in every extraction. Some of the online Eclipse forums attribute the timeout problem to heavy access of Eclipse servers during regular working hours. To circumvent this problem, we extracted change history during weekend nights and holidays.

Ying et al. do not indicate how they extracted history corresponding to the training period, to ensure that the history for all files as they existed on 2002/10/01 was extracted. Ying et al. did not use a date tag to extract history corresponding to 2002/10/01. Assuming that Ying et al. extracted the change sets from the main development branch, we selected a date on which we suspect the data may have been extracted by them. We repeated a variation of our experiment, described in detail in Section 2.3.

Next, we applied our modified FP-tree mining algorithm (using 10 as the frequent threshold value) on the extracted atomic change sets to obtain frequent patterns. Having extracted all frequent patterns for Eclipse, the next step involved picking 20 modification tasks and identifying their solution sets. Ying et al. do not give details about the solution set for each modification task. Instead, we used the heuristics as described by Ying et al. (i.e., use of the bug identifier in the check-in comment and a change in task status at the time of check-in) to identify solution sets. Neither their paper nor the thesis states if both heuristics must be satisfied to correctly identify the solution set or if satisfaction of one heuristic suffices. However, in our personal correspondence, Ying confirmed the simultaneous use of both heuristics to identify the solution set. For some of the tasks (described later), however, the first heuristic (use of the bug identifier in the check-in comment) did not result in any files. Though the second heuristic is useful to give more credence to the files already identified by the first heuristic, it is not sufficient to conclusively identify the solution set. This is because a developer typically performs several check-ins with different files in a couple of hours. Without an indication of "how close" the check-in should be with respect to the time when the status of the task is changed to "fixed" in Bugzilla, it was difficult to re-establish the solution set.

Nevertheless, we used the heuristics described by Ying et al. to identify solution sets for the modification tasks. We used the identified solution set even if one of the two heuristics was satisfied. In addition, we also used Ying et al.'s task descriptions from their journal publication and Ying's MSc thesis to corroborate our findings. In Table 2.2 we describe the details of solution sets for modification tasks based on the two heuristics. Columns "Heuristic 1 (dev. comments)" and "Heuristic 2 (check-in time)" show the heuristics that were satisfied in identification of the solution set. Note that in the absence of a well-defined satisfaction criterion for Heuristic 2, we only show the difference in time from the check-in time to the time when the status of the task was changed to "fixed". We now discuss the tasks where the solution set was identified somewhat differently than what Ying et al. suggest.

1. Bugs #21330 and #23587 did not satisfy the first heuristic. For these two bugs we used the details in Ying's MSc thesis to identify the solution set where she describes parts of the solution set for Bug #21330. In addition, we used the second heuristic to find the time when the bug was fixed in Bugzilla. The resultant solution set was used for this task.

2. Bugs #24635, #24657, and #23096 had a significant time difference (more than a day) between the solution check-in and Bugzilla "fix" of the task. For these tasks the solution set identified by only the first heuristic was used.

3. Bugs #24668 and #24756 returned only one file for the solution set and hence were not suitable for our experiments.

2.3 Results

Ying et al. used their approach to make predictions for 20 randomly selected Eclipse tasks. Of these 20 tasks, for two tasks we found only one file changed as part of the solution, making the solution set of cardinality one and hence ineffective for our experiments. We report these tasks as "N/A" in the results tables. Ying et al. report results for only nine out of 20 tasks. For the other 11 tasks, Ying et al. report that none of the files in the solution set were part of a change pattern.

We first repeat Ying et al.'s experiment for the 11 tasks that did not result in any recommendations. Ying states in her thesis [42, p. 39]:

> We could not provide recommendations for 11 of the modification tasks (#16817, #24165, #24406, #24424, #24449, #24594, #24622, #24756, #24828, #25124, and #25133) because the changed files were not covered by a change pattern.

Bug ID	Heuristic 1	Heuristic 2
	(dev. comments)	(check-in time)
24635	yes	+1d 6hr
21330	no	+2hr
24657	yes	-2d 22hr
25041	yes	0hr
13907	yes	+1hr
23096	yes	-1d 18hr
24668	yes	+2hr
23587	no	+2hr
24730	yes	+2hr
16817	yes	+2hr
24165	yes	+2hr
24406	yes	+1hr
24424	yes	+2hr
24449	yes	+15hr
24594	yes	+2hr
24622	yes	+1hr
24756	yes	+2hr
24828	yes	+2hr
25124	yes	+1hr
25133	yes	+2hr

Table 2.2: Heuristic details for identification of each modification task solution. The '+' sign indicates that the solution set for the task was checked in before the status of the task was changed to "fixed" in Bugzilla and '-' sign indicates the time for Bugzilla status change preceded the check-in time of solution set.

However, our experiments resulted in recommendations for 10 of these tasks, some of which were correct recommendations (see Table 2.3). We found a single file as a solution set for Bug #24756 and hence we do not have any results corresponding to this task.

Bugs #24406, #24449, #24622, and #25124 resulted in correct recommendations, as opposed to no recommendations as reported by Ying et al. On manual inspection, we found a common file in the solution sets of these four modification tasks. Each of the four tasks with correct results involved a build-notes file[4] in the solution set. This file had several hundred

[4]Named `org.eclipse.jdt.core/buildnotes_jdt.core.html`.

Bug ID	Recommendations	
	correct	incorrect
16817	0	6
24165	0	4
24406	2	30
24424	0	3
24449	2	30
24594	0	1
24622	2	30
24756	N/A	N/A
24828	0	5
25124	2	36
25133	0	1

Table 2.3: Experimental results for the 11 modification tasks in which Ying et al. report that no results occurred.

changes to it over time, as this file is used to record notes for each build. Also, the build-notes file changed with many other files and its longest frequent patterns involved 30 other files. Thus, when used as an input the build-notes file resulted in a high number of recommendations, few of which were correct.

When we removed the build-notes file from the solution set, all four tasks which had previously resulted in correct recommendations returned only incorrect recommendations (see Table 2.4). All the modification tasks now returned either incorrect recommendations or no recommendations. Later, in our teleconference meeting, Ying confirmed the removal of documentation files form the change history. However, the filtering of documentation files is not reported in their thesis or journal publication.

Our results still differ from Ying et al.'s, as our experiments resulted in recommendations for most tasks even after filtering out the build-notes file.

Next, we repeat Ying et al.'s experiment for the remaining nine tasks which had resulted in correct recommendations in their original experiment. For each of these nine tasks, Ying et al.

Bug ID	Recommendations	
	correct	incorrect
16817	0	6
24165	0	4
24406	0	2
24424	0	3
24449	N/A	N/A
24594	0	1
24622	N/A	N/A
24756	N/A	N/A
24828	0	5
25124	0	8
25133	0	1

Table 2.4: Experimental results for the 11 modification tasks, in which Ying et al. report that no results occurred, after filtering out the build-notes file. Note that for Bugs #24449 and #24622, removal of the build-notes file left only one file in the solution set which could not be used to compare recommendations for correctness.

report useful recommendations when their approach was applied to these tasks. They report a precision of around $\sim$0.3 and a recall of $\sim$0.1 to 0.2 for these tasks.

Figure 2.2, taken from Ying et al.'s journal publication, shows the Eclipse recommendations categorized by interestingness value (which we do not consider in our work). In our experiments, we detected a result where the modification task identifier seems to be erroneous. We found that Bug #24657 is incorrectly reported as #24567 in the second row of Figure 2.2. In her thesis Ying has described all the tasks in Mozilla and Eclipse experiments that result in surprising or neutral recommendations, yet Bug #24567 is not described. In addition, in the description of Bug #24657, Ying reports two neutral recommendations ("... although each of the files have parts that have call relationships to the other file, resulting in two neutral recommendations" [42, p. 41]) which we do not see in Figure 2.2. Finally, Ying et al. report only nine tasks resulting in correct recommendations for Eclipse but we see ten tasks in Figure 2.2. Thus, we consider the entry for Bug #24657 to be a typographical error, and that Bug #24567

was actually intended[5].

In Figure 2.2, "Description" describes the relationship between the input file and the recommended file. "Modification task ids (and no. of recommendations)" gives the bug identifiers and their corresponding number of recommendations whose relationship with the input file is as given in the "Description" column. We are not certain of this interpretation as Ying et al.'s description for it is not clear [41, p. 580]:

> The number in parentheses beside each identifier is the sum of the cardinality of recommended sets, where each recommended set is the result of querying with a different $[F_s]$ associated with the modification task.

Two things are unclear here:

- How should multiple instances of the same identifier be interpreted? For example, Bug #25041 is present under the descriptions "distant inheritance", "method call dependence" and "direct inheritance".
- Is the sum being referred to obtained by adding the cardinality of the recommended sets of *every* file in the solution set?

Due to the ambiguity in the description of the number in parentheses, we use the following interpretation for its meaning: "The number in parentheses represents the total number of input-file/recommended-file pairs that satisfy the given relationship in the 'Description' column." Consequently, when an identifier occurs multiple times, the sum of the numbers in parentheses for each occurrence is equal to the sum of the cardinality of the recommended sets obtained by using every file in the solution set. For example, Bug #25041 occurs three times and the sum of the numbers in parentheses for each of the three occurrences is $12 + 2 + 8 = 22$. So, adding the size of the recommended sets obtained with each input file in the solution set should result in 22 appearing in the parentheses.

[5]In our teleconference meeting later, Ying confirmed the typographical error.

Interestingness	**Description**	**Modification task ids (and no. of recommendations)**
surprising	cross-platform/XML	24635 (230)
neutral	distant call dependence	21330 (2), 24567 (2)
neutral	distant inheritance	25041 (12)
obvious	containment	13907 (2), 23096 (2), 24668 (2)
obvious	framework	21330 (2)
obvious	same package with less than 20 files	21330 (2)
obvious	instance creation/being created	23587 (4)
obvious	method call dependence	21330 (2), 23587 (4), 24657 (2), 25041 (2)
obvious	direct inheritance	25041 (8)
obvious	interface-implementation	24730 (2)

Figure 2.2: Ying et al.'s results. [Table reproduced with permission from IEEE.]

Based on this interpretation, we compare our results (along with Ying et al.'s) for the nine modification tasks that returned correct results in Ying et al.'s experiments in Table 2.5. In Table 2.6 we report the results after filtering out the build-notes file.

Bug ID	Our recommendations			Ying et al.'s recommendations
	correct	incorrect	total	(total)
24635	208	26	234	230
21330	6	11	17	8
24657	4	39	43	4
25041	40	53	93	22
13907	4	36	40	2
23096	6	36	42	2
24668	N/A	N/A	N/A	2
23587	12	4	16	8
24730	2	4	6	2

Table 2.5: Experimental results for the 9 tasks in which Ying et al. report results.

Table 2.7 summarizes the precision and recall values for the nine modification tasks. Table 2.8 summarizes the precision and recall values for the nine modification tasks without the

Bug ID	Our recommendations			Ying et al.'s recommendations
	correct	incorrect	total	(total)
24635	208	26	234	230
21330	6	11	17	8
24657	4	39	43	4
25041	24	39	63	22
13907	2	8	10	2
23096	2	10	12	2
24668	N/A	N/A	N/A	2
23587	12	4	16	8
24730	2	4	6	2

Table 2.6: Experimental results, for the 9 tasks for which Ying et al. report results, after filtering out the build-notes file.

build-notes file.

We found that our precision and recall rates differed widely from those reported by Ying et al. They reported a precision of around $\sim$0.3 and a recall of $\sim$0.1 to 0.2 [41, p. 580], while we obtained precision and recall rates of 0.5032 and 0.2877 respectively (0.48 and 0.27 after filtering out the build-notes file). Ying et al. only report average precision and recall rates making it difficult to identify for which individual tasks our results differ.

It is important to note that in our experiments we could not establish a solution set for Bug #24668 which we report as "N/A" in Tables 2.5, 2.6, 2.7, and 2.8. However, even if we assume the precision and recall rates for Bug #24668 as zero, average precision and recall rates for our experiments would still be 0.447 and 0.255, differing significantly from that of Ying et al.

As our results differ from Ying et al.'s both in number of recommendations and precision and recall rates we examine the results for a single modification task in detail. We examine one of the modification tasks for which the total number of recommendations in our experiment and Ying et al.'s experiments differed widely. We picked Bug #24657 which is a good candidate for

Bug ID	Our results	
	pr_m	rc_m
24635	0.8843	0.4127
21330	0.3333	0.1429
24657	0.1654	0.1600
25041	0.5952	0.1100
13907	0.2611	0.3333
23096	0.3292	0.2143
24668	N/A	N/A
23587	0.8571	0.4286
24730	0.6000	0.5000
Average	0.5032	0.2877

Table 2.7: Precision and recall results for the 9 tasks for which Ying et al. report results.

detailed inspection since the solution set for it could be established, unequivocally, based on the heuristics. Moreover, Bug #24657's solution set and experimental results are small enough to be examined carefully and manually.

To confirm the validity of our results we checked the results for Bug #24657 at each stage; its solution set consists of six files. First, we looked at the history of each file in the solution set to confirm that the file was involved in the solution by checking for the occurrence of "24657" in the log comments for the file. Next, we checked the correctness of our extracted change sets by manually checking two randomly selected change sets in the Eclipse CVS (containing some of the six files) using the heuristics to detect change sets (similarity of author name, comment, and timestamp). Next, we checked the validity of the recommendations. Each file, when used as an input, produced several recommendations in our experiment. To confirm the validity of the returned recommendations we computed the number of times the recommended file was changed with the input file.

For this last step, we wrote a Python script that calculates the frequent patterns by definition. For any input, the script iteratively finds the change sets containing the input. Next, it orders

Bug ID	Our results	
	pr_m	rc_m
24635	0.8843	0.4127
21330	0.3333	0.1429
24657	0.1654	0.1600
25041	0.4595	0.0741
13907	0.3125	0.3333
23096	0.2083	0.1111
24668	N/A	N/A
23587	0.8571	0.4286
24730	0.6000	0.5000
Average	0.4776	0.2703

Table 2.8: Precision and recall results for the 9 tasks, for which Ying et al. report results, after filtering out the build-notes file.

the files in these change sets by the number of times they occurred with the input. If a given file occurred ten or more times with the input file in change sets, it is returned as a recommendation. In this manner, we have an alternative way to find frequent patterns and hence compare the recommendations provided by our implementation of the FP-tree mining algorithm. We used this alternative script on five of the 20 modification tasks. Each of the recommended files (using the alternative script) for the five modification tasks, matched our results exactly from our implementation of Ying et al.'s approach. We conclude that our implementation is thus correct.

We present the recommendations provided for Bug #24657 in Tables 2.9(a) & (b). The first column ("File name") gives the file used as input (in bold) followed by the files obtained as recommendations. The second column ("# Recomm.") gives the total number of recommendations for the input file. The last column ("# Co-changes") gives the frequency of the number of times the recommended file changed with the input file. Based on our evaluation of Bug #24657 we believe that our results are correct; however, we still see a significant differ-

ence in our results compared to those of Ying et al. They report only four recommendations for Bug #24657 while we obtained 43 recommendations, four of which were correct.

To evaluate the correctness of our implementation of Ying et al.'s work we performed one last detailed test on a subset of recommendations for the Bug #13907. We selected Bug #13907 and manually tested the number of recommendations for one of the inputs. The rationale behind this last test was to eliminate the possibility of error by performing the entire life cycle of Ying et al.'s approach without using any component of our implementation. We only relied on the information provided by CVS to test the validity of the selected recommendations.

In Figure 2.2 Ying et al. report that Bug #13907 only produced 2 recommendations. However, we obtained 10 recommendations (2 correct, 8 incorrect). One of the input files `Parser.java` alone returned 8 files as recommendations (while Ying et al. report the *sum of recommendations* for all the inputs to be 2). Using Eclipse's WebCVS[6], a web interface to Eclipse's CVS repository where all the change log files for Eclipse can be accessed online, we retrieved all the log files for four of the eight recommended files (`Scanner.java`, `ProblemReporter.java`, `DocumentElementParser.java` and `SourceElementParser.java`). We then compared manually (using grep to search for relevant change log details) to see if these four files had changed at least 10 times with the input file, `Parser.java`. All four files were changed at least 10 times with the input file. This established that for Bug #13907 the total number of recommendations should be 4 or more.

2.3.1 Differences in data extraction

One of the stages in Ying et al.'s approach that is open to different interpretations and hence differences in results, is the data extraction stage. As described earlier, Ying et al. extract Eclipse change history from CVS in the form of atomic change sets. For their experiment they have used the atomic change sets in the duration of 2001/04/28 to 2002/10/01 as training data;

[6]http://dev.eclipse.org/viewcvs/

File name	# Recomm.	# Co-changes
dir/internal/core/InstallConfiguration.java	**23**	
dir/configuration/IConfiguredSite.java		10
dir/core/Feature.java		16
dir/core/Site.java		16
dir/core/SiteManager.java		10
dir/core/Utilities.java		10
dir/internal/core/ConfigurationActivity.java		10
dir/internal/core/ConfigurationPolicy.java		41
dir/internal/core/ConfiguredSite.java		25
dir/internal/core/FeatureExecutableFactory.java		10
dir/internal/core/FeaturePackagedContentProvider.java		12
dir/internal/core/InternalSiteManager.java		29
dir/internal/core/messages.properties		13
dir/internal/core/SiteFile.java		12
dir/internal/core/SiteFileFactory.java		18
dir/internal/core/SiteLocal.java		61
dir/internal/core/SiteReconciler.java		12
dir/internal/core/UpdateManagerPlugin.java		12
dir/internal/core/UpdateManagerUtils.java		25
dir/tests/configurations/TestRevert.java		13
dir/tests/parser/TestSiteParse.java		10
dir/tests/regularInstall/TestInstall.java		14
dir/tests/regularInstall/TestInstallURLSIteXML.java		11
dir/tests/regularInstall/TestLocalSite.java		22

Table 2.9(a): Detailed results for Bug #24657, Part 1. Note that “dir” = /org/eclipse/update.

File name	# Recomm.	# Co-changes
dir/core/JarContentReference.java	**2**	
dir/core/FeatureContentProvider.java		14
dir/internal/core/UpdateManagerUtils.java		10
dir/core/FeatureContentProvider.java	**12**	
dir/core/ContentReference.java		12
dir/core/Feature.java		18
dir/core/IFeature.java		10
dir/core/IFeatureContentProvider.java		11
dir/core/JarContentReference.java		14
dir/core/Site.java		12
dir/internal/core/FeatureExecutableContentProvider.java		10
dir/internal/core/FeaturePackagedContentProvider.java		19
dir/internal/core/messages.properties		10
dir/internal/core/SiteFile.java		10
dir/internal/core/SiteFileFactory.java		10
dir/internal/core/UpdateManagerUtils.java		11
dir/internal/core/UpdateManagerLogWriter.java	**0**	
dir/core/Utilities.java	**5**	
dir/core/Feature.java		10
dir/internal/core/ConfiguredSite.java		10
dir/internal/core/InstallConfiguration.java		10
dir/internal/core/InternalSiteManager.java		10
dir/internal/core/SiteLocal.java		12
dir/core/FeatureReference.java	**1**	
dir/core/Feature.java		10

Table 2.9(b): Detailed results for Bug #24657, Part 2. Note that "dir" = /org/eclipse/update.

naturally, the actual extraction took place after 2002/10/01. To extract the history of all files existing in the Eclipse code base on a certain date, CVS logs should be extracted against the date tag for that particular date. However, Ying et al. do not elaborate on their process for atomic change set extraction. One possibility is that Ying et al. have extracted the history from the current development branch and later filtered it to include only history within the training data time period; this approach would not affect the change sets involving files existing in the system for the duration of training data and current development branch; however, the history for files existing during training data but deleted from the current development branch would be missed. To test this hypothesis, we re-extracted the history differently from our original experiment. We re-extracted the atomic change sets for the code base as existing on 2003/06/30, which we suspect to be the approximate date of extraction by Ying et al.

Table 2.10 shows the results of the variation of experiment based on the date tag. It reports the results for the nine modification tasks with useful results after filtering out the build-notes file. The resulting experiments returned results which were similar to our original results (see Table 2.6), with moderately to significantly higher numbers of recommendations than those of Ying et al. It is possible that the Eclipse code base, or more precisely the set of files returned as recommendations in our experiments, did not undergo the kinds of major additions or deletions during the training data time period necessary to differentiate these two extraction methods.

2.3.2 Differences in computing average precision and recall

After checking various hypotheses explaining the differences in results, we realize that there could be an alternative interpretation of Ying et al.'s description for average precision and recall values.

We interpret their description of average precision and recall to be the mean of pr_m and rc_m values *for each modification task*, as described above. However, an alternate interpretation could be that average precision and recall are the mean precision and recall values *over all*

Bug ID	Our recommendations			Ying et al.'s recommendations
	correct	incorrect	total	(total)
24635	206	26	232	230
21330	2	6	8	8
24657	4	38	42	4
25041	20	36	56	22
13907	2	8	10	2
23096	2	10	12	2
24668	N/A	N/A	N/A	2
23587	12	4	16	8
24730	2	4	6	2

Table 2.10: Experimental results for variation based on date tag for 9 modification tasks, for which Ying et al. report results, after filtering out the build-notes file.

inputs, regardless of task. To better differentiate this new interpretation of the final mean values, we refer to them as $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$, as defined in Equations 2.7 and 2.8.

$$\mathrm{pr}'_{\mathrm{avg}} = \frac{1}{\sum_{m \in M} |F_s(m)|} \sum_{m \in M} \sum_{f_s \in F_s(m)} \mathrm{pr}_m(\mathrm{Rec}(\{f_s\}), F_s) \tag{2.7}$$

$$\mathrm{rc}'_{\mathrm{avg}} = \frac{1}{\sum_{m \in M} |F_s(m)|} \sum_{m \in M} \sum_{f_s \in F_s(m)} \mathrm{rc}_m(\mathrm{Rec}(\{f_s\}), F_s) \tag{2.8}$$

Thus, for threshold 10 for the 9 modification tasks for which Ying et al. report results, we obtain $\mathrm{pr}'_{\mathrm{avg}} = 0.324$ and $\mathrm{rc}'_{\mathrm{avg}} = 0.127$. Note that, if we include the precision and recall values for the remaining 11 modification tasks, then we get the following values: $\mathrm{pr}'_{\mathrm{avg}} = 0.243$ and $\mathrm{rc}'_{\mathrm{avg}} = 0.11$. These values (for the 9 modification tasks) match the ones reported by Ying et al. in their journal publication: precision of around ∼0.3 and a recall of ∼0.1 to 0.2. Thus for the 9 modification tasks for which Ying et al. obtained valid recommendations, we have the same results as them. However, we obtained valid (albeit incorrect) recommendations for some of remaining 11 tasks for which Ying et al. report that they did not obtain any recommendations.

We feel that although we deviated from Ying et al.'s method of calculating $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$, our intermediate measures of $\widetilde{\mathrm{pr}_m}$ and $\widetilde{\mathrm{rc}_m}$ provide additional insight in the usefulness of rec-

ommendations. More precisely, $\widetilde{\mathrm{pr}_m}$ and $\widetilde{\mathrm{rc}_m}$ are useful in indicating if a recommender approach performs better for certain types of modification tasks. We discuss $\widetilde{\mathrm{pr}_m}$ and $\widetilde{\mathrm{rc}_m}$ in more detail in Chapter 3.

From now on in this thesis we use $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$ and an additional measure of $\mathrm{tp}'_{\mathrm{avg}}$ (introduced in Subsection 2.3.3) as our primary measures for evaluating the efficacy of a recommender approach.

2.3.3 Throughput

As mentioned in Section 2.2.1, the individual inputs that do not return any recommendations are assigned precision and recall values of 0 each. Ying et al. did not discuss this case for individual inputs and while assigning 0 for both precision and recall is one way to count the values, it can introduce a bias in the results.

To elaborate, imagine a scenario in which 50% of inputs result in incorrect recommendations and a parallel scenario where the same 50% of the inputs result in no recommendations. While the second scenario is much more favourable (no results are better than misleading results), our current way of counting precision and recall values for inputs with no recommendations would make it impossible to distinguish between the first and the second scenario.

We believe that ignoring inputs without recommendations from the computation of Equations 2.7 and 2.8 and introducing an additional measure of "throughput" to depict the fraction of inputs that resulted in recommendations (correct or incorrect) is a better way of capturing the overall usefulness of a particular approach. Zimmermann et al. [44] follow this approach to report the results for their CHB recommender approach. Following this technique, for threshold 10 we get the following new values: $\mathrm{pr}'_{\mathrm{avg}} = 0.486$, $\mathrm{rc}'_{\mathrm{avg}} = 0.22$ and $\mathrm{tp}'_{\mathrm{avg}} = 0.5$. Thus, henceforth in this thesis we use these three measures to capture the results of our experiments for the recommender approaches under consideration.

2.3.4 Finding the best threshold

Ying et al. perform their experiments for four different thresholds (5, 10, 15 and 20) [41, p. 579] and obtained the best results for threshold 10. To more accurately observe the effect of threshold on the quality of results we repeat the experiments for a range of thresholds. We found that after threshold 30 the throughput is negligible, and for threshold 1 the FP-tree mining algorithm is computationally very expensive and cannot be applied easily. Hence we select the threshold range to be tested as 2–30.

Figure 2.3 shows the effects of varying threshold on $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$. Different thresholds give the best values for each of the measures. This is expected as a higher number of recommendations not only implies more correct recommendations (increase in recall) but also more incorrect recommendations (decrease in precision). On the other hand, at higher threshold values, the number of recommendations decreases resulting in lower recall rates but variable precision rates (based on the decrease in correct or incorrect recommendations). In the end, the importance of higher precision versus higher recall depends on the use case. A developer looking to get as many relevant files as possible for an unfamiliar code base might be interested in higher recall values, while a developer looking for one last file to complete a bug fix would want fewer recommendations with higher accuracy (higher precision).

For our experiments we obtained the following best values:

1. Best value for $\mathrm{pr}'_{\mathrm{avg}} = 0.56$ at threshold 13 ($\mathrm{rc}'_{\mathrm{avg}} = 0.13$, $\mathrm{tp}'_{\mathrm{avg}} = 0.31$)

2. Best value for $\mathrm{rc}'_{\mathrm{avg}} = 0.58$ at threshold 2 ($\mathrm{pr}'_{\mathrm{avg}} = 0.23$, $\mathrm{tp}'_{\mathrm{avg}} = 0.87$)

3. Best value for $\mathrm{tp}'_{\mathrm{avg}} = 0.87$ at threshold 2 ($\mathrm{pr}'_{\mathrm{avg}} = 0.23$, $\mathrm{rc}'_{\mathrm{avg}} = 0.58$)

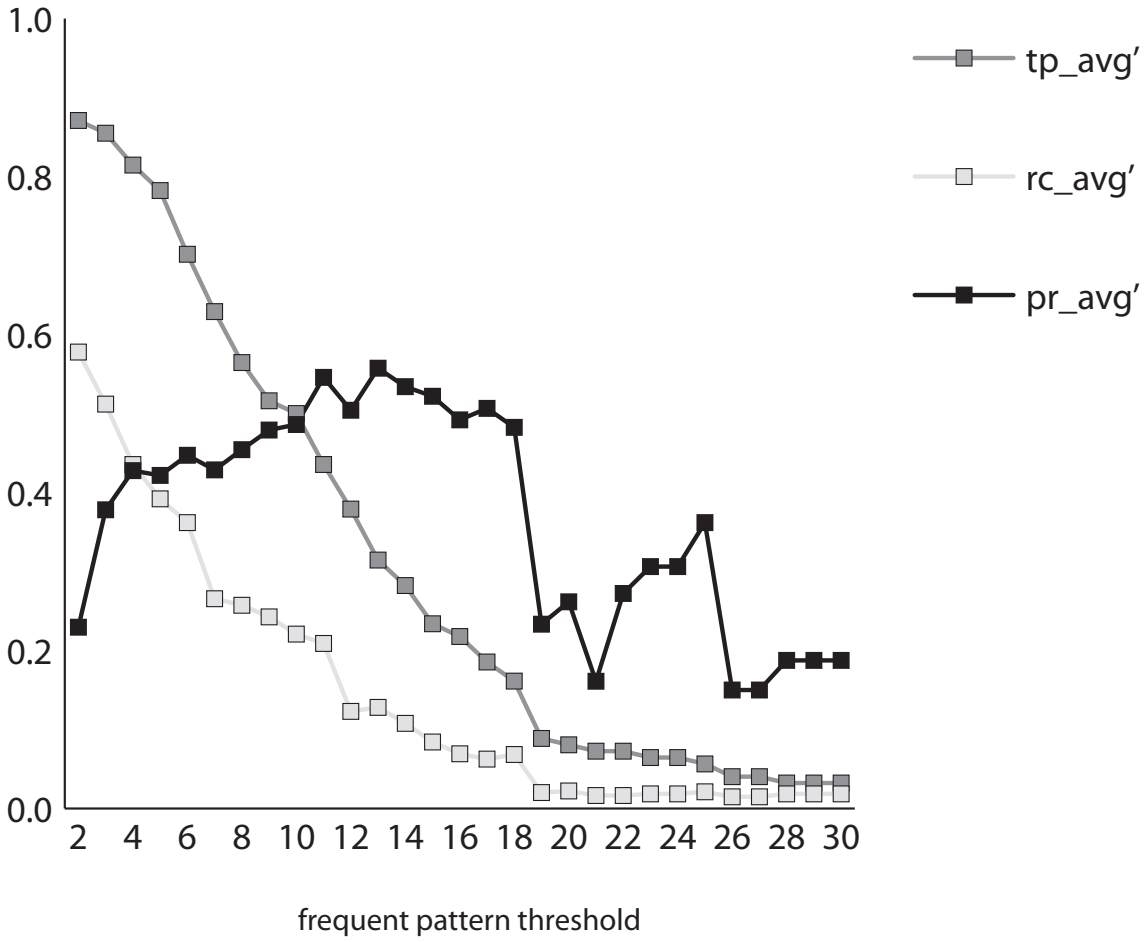

Figure 2.3: Effect of different thresholds on $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$ for Ying et al.'s approach at file-level granularity.

2.4 Summary

In this chapter, we describe our re-implementation of Ying et al.'s change history based recommendation approach and our partial replication of their experiment for their change history mining recommender approach.

We had difficulties replicating their experiment due to the ambiguity of a number of details:

Data pre-processing. The implementation of atomic change set extraction and removal of spurious change sets is fairly straightforward based on Ying et al.'s description and information obtained through our personal correspondence. The only deviation in our understanding could be due to differences in the date at which data was extracted. To test for this difference we conducted a variation of the experiment using a different extraction date tag. The results did not differ significantly from our original experiments implying that the extraction date tag did not account for our results that differ from those of Ying et al.

Frequent pattern extraction. Though the implementation of the original FP-tree mining algorithm is straightforward, we were unable to determine a simple means to return only maximal patterns, which Ying et al. state they had accomplished. However, we believe that we interpreted the rationale behind the modifications correctly and achieved the same results as Ying et al. at a later stage in the experiment. We also validated the correctness of our modified FP-tree mining algorithm by implementing an alternate simpler program to find recommendations (the same as those provided by the frequent pattern technique)

Experiments. We found that the actual experimental setup and quantitative measures of results to be the most troublesome issues. The three major sources of potential deviations from their work were finding solution sets for the modification tasks, counting the returned recommendations and calculating final $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ values. As we found the

heuristics used for identifying solution sets imprecise and hence subject to different interpretations, we used all the information available to us to find the solution sets and explain the variations where applicable. Lacking quantitative details for the results meant that we could not find the exact cases where our results differed with that of Ying et al.'s leaving us to compare only cumulative results. However, in the end we believe that the differences in results were in large part due to differences in calculating average precision and recall (i.e., $\mathrm{pr}_{\mathrm{avg}}$ and $\mathrm{rc}_{\mathrm{avg}}$ vs. $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$).

In the 11 modification tasks for which Ying et al. report no recommendations, our experiments resulted in recommendations, some of which were correct (i.e., true positives). We observed that for all these tasks where we obtained correct recommendations, a build-notes file was involved with a very long frequent pattern. The file itself is presumably modified frequently to record notes about changes, and is not part of modification task solutions as such. We removed the file from the solution set and the new results were closer to Ying et al.'s with no correct recommendations (we still obtained some incorrect recommendations rather than no recommendations).

Next, we repeated the experiments for the 9 modification tasks that resulted in correct recommendations for Ying et al.'s experiments. Our experiments resulted in significantly higher numbers of recommendations compared to those reported by Ying et al. Our precision and recall values differed too (0.503 and 0.287 for precision and recall, respectively, as opposed to 0.3 and 0.1–0.2 reported by Ying et al.). We repeated the experiments after filtering out the build-notes file; this did not change the results by much (0.477 and 0.27). However, removal of the build-notes file did result in lower numbers of recommendations that were closer to those of Ying et al.'s results.

In the end, we realized that there might be a different interpretation of Ying et al.'s description of average precision and recall. Following an alternate technique for combining precision and recall values for individual inputs we arrived at the same average values as reported by

Ying et al. However, it is important to take into account that our results still differed for some of the remaining 11 tasks for which Ying et al. did not report any recommendations.

Ultimately, we can only compare our results against the cumulative precision and recall values as reported by Ying et al. for the 9 modification tasks, with any certainty. The number of recommendations, though more detailed in their publications, are open to interpretation making it difficult to compare our results. Nevertheless we performed two variations of our experiments, one of which changed the number of recommendations bringing them closer to Ying et al.'s (removal of the build-notes file), the other not affecting the results by much (variation based on date tag).

Last, we repeated their experiments for several more thresholds (in addition to the 4 thresholds used by them in their original experiments). We also introduced a new measure "throughput", in addition to altering the original $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ measures to better represent the overall results. We found that if we pick the frequent threshold with best $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ values (using either their original technique or our new measures), we would actually get much better $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ values for threshold 5 ($\mathrm{pr}'_{\mathrm{avg}} = 0.42$, $\mathrm{rc}'_{\mathrm{avg}} = 0.39$, $\mathrm{tp}'_{\mathrm{avg}} = 0.78$).

To conclude, we believe that we have reasonably evaluated the correctness of our implementation of their approach. While our interpretation of some details of their approach and experiment may not accurately reflect what they did, we believe that we have demonstrated that their approach actually works somewhat *better* than they have reported.

Chapter 3

YING ET AL.'S APPROACH FOR METHOD-LEVEL GRANULARITY

A crucial consideration when designing a recommender system is the intended granularity of the results. Selection of any granularity comes with its inherent tradeoffs. Finer-grained recommendations might provide more pertinent results while coarser-grained recommendations might capture a broader scope in its results. To clarify, imagine two extremes: a very fine grained recommendation could suggest particular lines of code, or individual source code expressions, to the developer. Such a fine-grained recommendation, though more useful to the developer, is much harder to achieve. On the other hand, a recommender system suggesting an entire project to a developer working on a modification task within it will always be "correct", but of little use to the developer. Ultimately, the factors affecting choice of granularity for a recommender system depend on the use cases for the recommender system, the availability of quality data, and the cost (in terms of developer's time, computational resources, etc.) of providing finer-grained recommendations.

Ying et al. implemented their change history based (CHB) recommender approach at a file-level granularity, i.e., their approach provides recommendations in the form of source code files.[1] However, Ying et al. raise the issue of granularity of their approach [41, p. 583]:

> Currently, the change associations we find are among files. Applying our approach to methods where change patterns describe methods instead of files that change together repeatedly may provide better results because a smaller unit of source code may suggest a similar intention behind the separated code. However, refining

[1] In Java, the name of a file containing a public, outer class is the same as the name of the class.

the granularity weakens the associations (each pattern would have lower support), which may not be well-handled by our current approach.

To investigate if Ying et al.'s CHB approach and ultimately our extended implementation thereof (extended CHB, or ECHB, approach) work better for a finer granularity, we also implement Ying et al.'s original approach for method-level granularity. In this chapter we describe this implementation (Section 3.1), and our modification for method-level granularity of their experiment (Section 3.2). Finally, we compare and contrast the results of Ying et al.'s approach at method-level and file-level granularities (Section 3.3).

3.1 Implementation

In this section, we briefly summarize our implementation of Ying et al.'s approach at method-level. Some variations from their approach are inevitable when porting it to a finer granularity; we discuss these in detail.

Ying et al.s original CHB recommender approach consists of three stages:

1. data preprocessing
   (a) identifying atomic change sets
   (b) filtering
2. association rule mining
3. query

Most of our variations occur in the first stage, with fewer in the second stage, and none in the last stage.

Data preprocessing, the first stage, involves extracting the change history of a system in the form of atomic change sets and filtering out any spuriously long change sets. In their

original approach, Ying et al. identify atomic change sets based on metadata associated with each source code transaction in a Version Control System. Thus files checked in together by the same author, using the same comment, within three-minute interval of each other are considered to form an atomic change set.

To port this approach to the method-level, we extracted the change history for a given system in the form of *method-level change sets*. This implies that the change sets must consist of methods (in place of files) that were changed by the same author, using the same comment within three minute interval.

Current Version Control Systems, like CVS, store the code in the form of text files. Each file in turn is associated with a set of revisions that are stored in the form of lines changed. Thus, CVS is unaware of changes at the syntactic level. So, to detect changes at the method level, we must map physical changes (files and line numbers) to syntactic changes (method changes). For this mapping, we included an additional step in Ying's data preprocessing stage.

In this step, we first extract the history of each source code file in the system in the form of revisions. A file revision is a snapshot of the file contents at a given point in the history of the file along with metadata such as revision number of the file, author of the revision, comment associated with the revision and timestamp of the revision. Next, we identify the syntactic entities that have changed in a given revision (compared to the previous revision). Determining the syntactic entities that have changed in a given revision is non-trivial. A naive comparison of two revisions of a file will provide only textual changes.

For example, Figure 3.1 shows the difference between CVS diff output and the syntactic differences for two revisions of a sample file, `Shape.java`. CVS stores the changes in the form of physical lines like `this.x = x;` (line #22 in `Shape.java`, revision 1). Each added line is denoted by a plus symbol and a deleted line denoted by a minus symbol. Our approach requires changes in the form of entities (more specifically, methods). Thus we must map changes in the form of line numbers (deleted lines #21–24 in `Shape.java`, revision 1) to

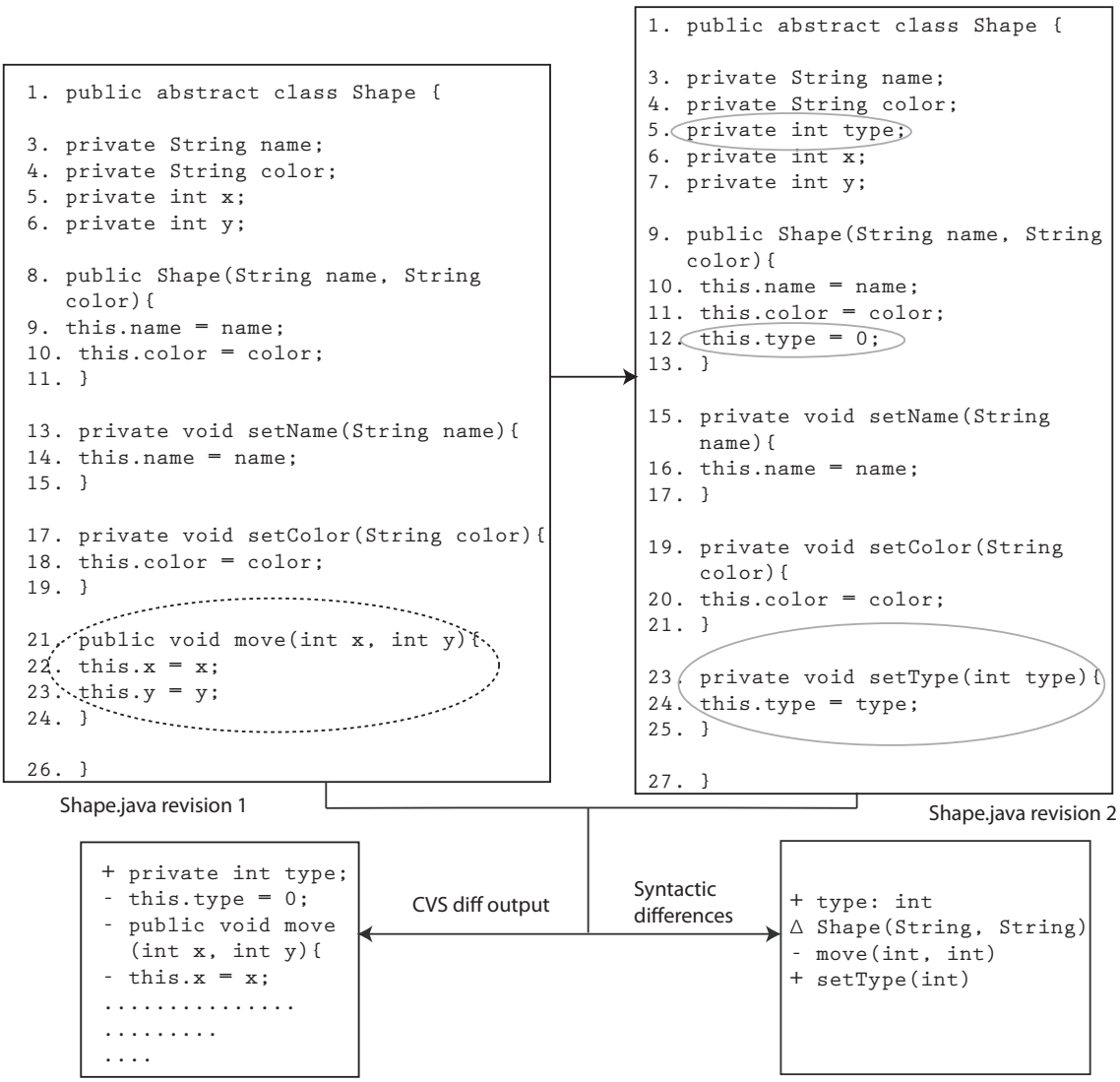

Figure 3.1: Mapping line number differences to syntactic differences.

syntactic changes, e.g., "method `move(int, int)` is deleted".

To map text changes to syntactic changes, we first determine all the syntactic entities in the two revisions of any given file. The `org.eclipse.compare` subproject[2] of Eclipse provides support for the comparison of two structures (source code files in our case) and finding the differences between them at syntactic level.

To find the syntactic differences between two files, we first map the revisions to a Java

[2]We use revision 3.4.1.r34x_20090121 in our implementation.

programming language structure (Java Structure). We leverage Eclipse's internal class `JavaStructureCreator.java`[3] for the mapping. `JavaStructureCreator.java` superimposes a hierarchical Java structure on a document. Next we find the syntactic differences between two Java Structures (for two revisions) using Eclipse's `Differencer.java` class[4].

`Differencer.java` is a generic differencing engine which can be used to compare two versions of a given hierarchical structure. Clients can write their own implementation of the hierarchical structure that they want compared. In our implementation, the differencing engine takes as input two revisions of a Java file in the form of Java Structures (obtained earlier) and returns a tree of differences. The inputs are hierarchical tree structures representing the Java files. The top most element of the structure is the class itself. Elements in the structure can have child elements of their own. The children are essentially syntactic entities which in turn may have more children. For example, a Java class is a parent of all its inner classes, each of which, in turn, is a parent of all its syntactic entities (methods, variable declarations etc.). Elements with no children form the leaf nodes of the structure.

The traverse operation of the engine, which results in a tree of differences, takes three inputs (the root node of the tree of differences (`inputNode`), `leftInput`, and `rightInput`) and performs a compare on the input objects using the following steps:

```
traverse(inputNode, leftInput, rightInput)
```

1. Create a new child node, such that its parent node is the `inputNode`. Annotate the newly created node with details of `leftInput` and `rightInput`.

2. Enumerate all the children for the two inputs. Store the left input's children as `leftSet` and right input's children in `rightSet`. Store the union of set of children for left and right input in an `allSet`.

3. For each child entity in the `allSet`, extract the corresponding entity (by the same name)

[3] `org.eclipse.jdt.internal.ui.compare.JavaStructureCreator.java`
[4] `org.eclipse.compare.structuremergeviewer.Differencer.java`

from `leftSet` and `rightSet`.

4. The two extracted child entities form new inputs for the `traverse` method of the differencing engine. Thus a recursive call to traverse is made, with left and right child entities as left and right inputs and the current node as parent node.

5. Steps 2-4 are repeated if both the `leftInput` and `rightInput` have children.

   (a) If one of the two do not have children we have reached an entity that exists only in one of the two inputs. Implying an `ADDED` or `DELETED` entity (`ADDED` if the entity only exists in the `rightInput` and `DELETED` if the entity only exists in the `leftInput`).

   (b) If both `leftInput` and `rightInput` do not have children we have reached a leaf node which can now be compared by invoking a byte-wise compare operation on the source code for the two entities. The compare operation returns a boolean value indicating a `CHANGE` or `NO_CHANGE`.

Thus, the extracted differences belong to one of the following categories:

1. `NO_CHANGE`: Method exists in versions n and $n-1$ without any change;

2. `CHANGED`: Method exists in versions n and $n-1$ but do not match;

3. `ADDED`: Method does not exist in version $n-1$ but exists in version n; and

4. `DELETED`: Method exists in version n but not in version $n-1$.

For simplicity, we consider all three types of differences (`CHANGED`, `ADDED`, `DELETED`) as a change.

Next, we create atomic change sets from the method-level changes, based on the closeness of changed methods in time using file revision metadata as implemented by Ying et al. (described in Chapter 2). To avoid any spuriously long change sets caused by activities such as

merging, we remove all change sets containing more than 400 methods for the method-level implementation of Ying et al.'s approach (for their file-level approach, Ying et al. removed all change sets containing more than 100 files). These large sets usually do not correspond to meaningful changes and add to the computational complexity by creating long branches in the frequent pattern tree during mining. Exceptionally large change sets result in a dense tree with more patterns and/or longer patterns resulting in expensive mining of frequent patterns.

Note that for non-Java files (i.e., text files, XML files, etc.) we do not extract changes at a finer granularity. Currently, our approach employs the knowledge of hierarchical structure of Java source code entities to determine fine-grained changes. Inclusion of files of a different nature would require us to define the structure for file types that do not have a native structure (for example, text files) and identifying the structure of files types that do have a native structure (for example, XML files). Our similarity-based approach for detecting transformation should work for any file type for which we can define fine-grained entities and a way of comparing them. However, the addition of each file types for this approach is non-trivial [10].

We summarize the method-level atomic change set creation for Java files through the following steps:

1. For a given Java file, all its revisions are extracted in the form of text files.

2. In addition, metadata for each revision is extracted (author name, comment, time stamp) which would be used later in constructing atomic change sets.

3. Next, each revision for a given Java file, from the most recent to least recent, is compared with its preceding revision to identify the structural changes. This is done by first mapping each revision to a Java Structure. The two revisions are compared using Eclipse's `Differencer` class.

4. The `Differencer` class returns the outcome of comparison (for two file revisions) in the form of a tree. Each leaf node in the tree represents a structural difference. We do

not distinguish between additions, deletions, and modifications.

5. All the method-level changes (along with their metadata) are now processed to form atomic change sets.

Our method-level implementation has two deviations from the file-level approach for the detection of changes (#3 and #4 in the steps enumerated above).

- In the file level approach, even the initial commit is recorded as a "change". However, as the initial commit cannot be compared with a preceding revision we do not include it as a change for the method-level implementation. To maintain consistency, for the method-level implementation, we do not record initial commits as changes for non-Java files either.

- Another deviation from the file-level approach is that, while deleted revisions still result in "changes" for the file-level approach, they are not included for the method-level approach. A deleted revision implies that all the contents of a file for that particular revision are deleted, however, CVS still retains the metadata associated with the deleted revision.

Both the deviations from the file-level approach stem from the fact that for method-level implementation of Ying et al.'s approach we require the source code for two revisions to generate method-level changes. In the first deviation, for the first commit, there is no "previous revision" to compare against. We believe that ignoring initial check-ins can lead to a lowered number for the best frequency threshold. For example, in our approach a method that has changed 5 times would be recorded as if it changed only 4 times. This scenario can also be extended to frequent patterns with multiple methods. Thus if methods a, b, and c changed 5 times, one of the changes stemming from the first commit, this frequent pattern would be recorded with frequency 4 in place of 5. Together such cases can bring down the best frequency threshold by 1. However, as we repeat our experiments for a range of thresholds, readers can observe the

effect of different thresholds without being misled by the results from a single best threshold. In addition, in this thesis we do not prescribe a best threshold but present the results for a range thresholds further eliminating the issue of incorrect best threshold.

For the second deviation, CVS stores the metadata even for a deleted file revision, which suffices for generating file-level changes. However, the source code, in the form of text files, for a deleted revision is not easily retrievable, making it complicated to generate changes for method-level implementation for deleted revisions. The number of missing changes due to deleted revisions, constitute only 0.2% of the total number of changes and as such should not affect the results.

We show the difference between the atomic change sets at two different granularities in the following example. Changes to non-Java files are represented in the same manner for both granularities.

File-level change set:

- `buildnotes.html`
- `spring.xml`
- `my/example/project/Process.java`
- `my/example/project/Shape.java`

Method-level change set:

- `buildnotes.html`
- `spring.xml`
- `my/example/project/Process.java/run()`
- `my/example/project/Shape.java/draw()`

- `my/example/project/Shape.java/Shape()`

The next stage uses the atomic change sets generated in the previous stage to extract frequent patterns using the frequent pattern mining algorithm (see Section 2.1.2). Our only deviation from Ying et al.'s approach in this stage is the user-provided thresholds that result in the most useful recommendations. Ying et al. experimented with various thresholds (5, 10, 15, 20) to determine the best threshold. They found 10 to be the best threshold resulting in relatively high values for both precision and recall. We repeat their approach and test the method-based implementation for a range of frequency thresholds, 2–20. For file-level granularity the range was 2–30; we have lowered the upper limit of the range from 30 to 20 to account for weaker associations due to finer granularity, as two specific methods have a lower probability of changing together than their respective container classes.

3.2 Experiment Replication

To assess the effect of granularity on the quality of recommendations we base our experiment on Ying et al.'s original.

We use 20 modification tasks from Eclipse that Ying et al. had randomly selected to test the effectiveness of their original approach. As described in Chapter 2.2, we identified some ambiguities in their experiment description during our replication of their original experiment for file-level granularity, for which we settled on the best interpretation. We utilize the same interpretations for our method-level implementation when unsure of the meaning or intent of Ying et al.'s description.

To identify the method-level solution sets for these modification tasks, we again repeat Ying et al.'s technique for identifying file-level solution sets. To identify the solution set for the tasks that were reported and fixed during the test data period (2002/10/01 to 2003/05/01) we created a CVS *date tag* for 2003/05/01 (corresponding to the end of test data) and extract

history relative to this date tag. This ensures that we get the history of the Eclipse code base as if the history was extracted from the main development branch on 2003/05/01. Section 2.2 describes our extraction procedure in detail.

Similarly, for the training data we extract the history corresponding to the *date tag* for the end of the training data time period (2002/10/01). We extract atomic change sets from this history and apply FP-tree mining algorithm on the atomic change sets to obtain frequent patterns.

Next, we use the two developer practices (use of task identifier in solution set check-in and the proximity in time of the solution set check-in with respect to change in status of the task in Bugzilla) leveraged by Ying et al. in determining the solution set for each modification task. The identification of solution set and ambiguities in resolving the solution set for certain modification tasks is described in Section 2.2. Thus, we identify the solution set for modification tasks for method-level granularity.

The following steps summarize our experiment for method-level granularity.

1. We extract method-level change sets for the Eclipse code base (everything in the package hierarchy for `org.eclipse.*`) against two date tags, one for the training data (2002/10/01) and one for the test data (2003/05/01)[5].

2. We use the atomic change sets from the training data to extract frequent patterns for different thresholds.

3. We identify the solution set for the modification tasks fixed within the test data time period using the heuristics described by Ying et al.[6] We filter out the build-notes file from the solution set as the file is a generic documentation file and skews the results (the recommender approach expects source code files).

[5]See Section 2.2 for variations in data extraction
[6]See Section 2.2 for variations.

4. We use *each* method in every solution set (identified as described above), for a given modification task, as an input for recommendations. All frequent patterns matching the input are extracted and all unique methods in the matched the frequent patterns are returned as recommendations.

We now report the results for the method-level implementation of Ying et al.'s approach for different thresholds. To identify the best threshold for method-level granularity we vary the frequency threshold from 2 to 20 and record the results for each threshold. We use the measures of $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$ to assess the usefulness of recommendations.

Figure 3.2 shows the effects of varying threshold on $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$.

For our experiments we obtained the following best values:

1. Best value for $\mathrm{pr}'_{\mathrm{avg}} = 0.89$ at threshold 17 ($\mathrm{rc}'_{\mathrm{avg}} = 0.12$, $\mathrm{tp}'_{\mathrm{avg}} = 0.01$)

2. Best value for $\mathrm{rc}'_{\mathrm{avg}} = 0.41$ at threshold 10 ($\mathrm{pr}'_{\mathrm{avg}} = 0.74$, $\mathrm{tp}'_{\mathrm{avg}} = 0.03$)

3. Best value for $\mathrm{tp}'_{\mathrm{avg}} = 0.24$ at threshold 2 ($\mathrm{pr}'_{\mathrm{avg}} = 0.35$, $\mathrm{rc}'_{\mathrm{avg}} = 0.21$)

During our experiments we noticed that the solution set for Bug #25124 consisted of 421 methods. This was an unusually large solution set as compared to the remaining bugs. In fact, the solution set for Bug #25124 (421 methods) was much larger than the solution sets for all the other bugs combined (281 methods). Figure 3.3 shows the relative sizes for the solution sets for the 20 bugs.

We observed that, even for lower thresholds, no recommendations were returned by most methods in the solution set for Bug #25124 when used as an input. For example, for threshold 3 only 8 out of 421 methods in the solution set when used as an input returned valid recommendations. Naturally, this resulted in very low $\mathrm{tp}'_{\mathrm{avg}}$ value for Ying et al.'s method-level approach. This brings us to an important observation, that our current approach of calculating $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$ (based on Zimmermann et al.'s [44] approach and similar to Ying et al.'s [41]) does

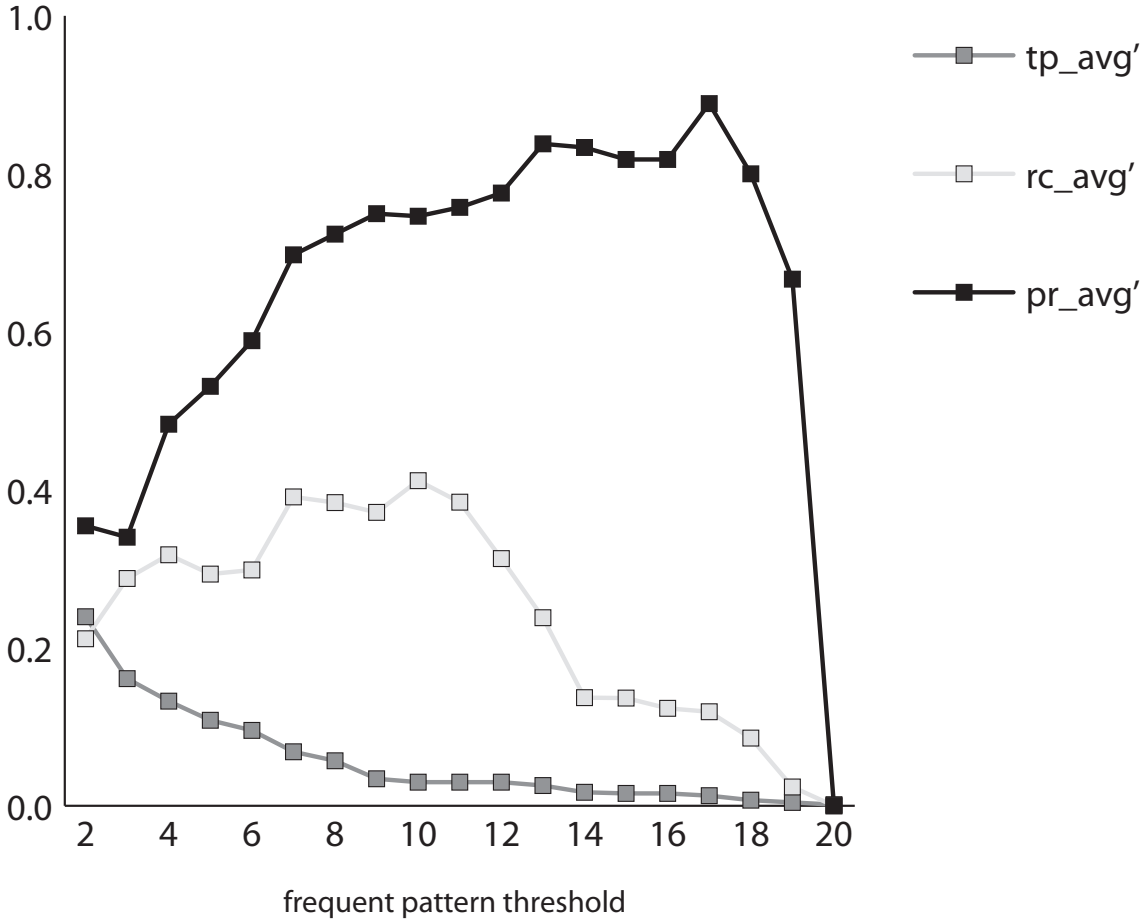

Figure 3.2: Effect of different thresholds on $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$ for Ying et al.'s approach at method-level granularity.

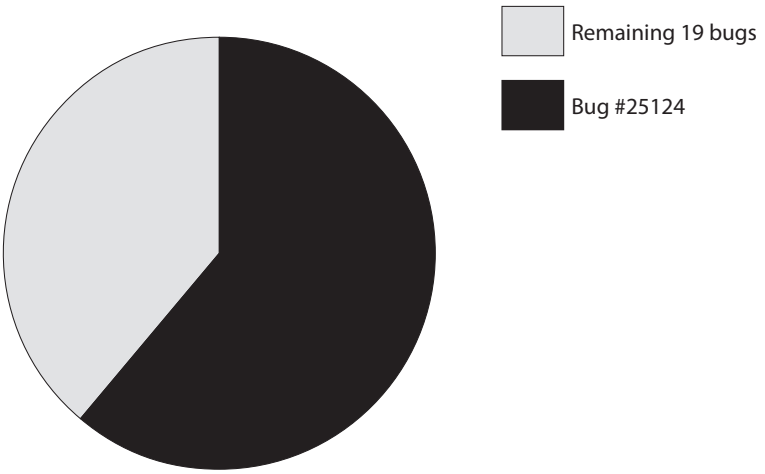

Figure 3.3: Solution set size (method-level granularity).

not give any indication of particular bugs for which the approach fails (or works better). The current approach is also biased towards bugs with larger solution sets since it assigns the equal weight to each input. Thus, the more inputs for a particular bug, the greater its influence on the overall results.

However, based on 20 bug fixes for which we performed the experiments, it appears that Ying et al.'s approach might be well-suited for bug fixes involving a smaller number of files (or methods) in the solution set. To observe the usefulness of Ying et al.'s approach for individual bug fixes we report our results in an alternate way. We report the $\widetilde{\mathrm{pr}_m}$ and $\widetilde{\mathrm{rc}_m}$ rates for each modification task and then find the mean of these averages. We refer to these as $\mathrm{pr}_{\mathrm{avg}}$ and $\mathrm{rc}_{\mathrm{avg}}$ to distinguish them from $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$. Note that, initially during partial recreation of Ying et al.'s approach we were reporting precision and recall values task-wise and used the notations $\widetilde{\mathrm{pr}_m}$ and $\widetilde{\mathrm{rc}_m}$ given by Equations 2.3 and 2.4 respectively. However, for inputs with no recommendations we were assigning 0 for precision and recall, now, on the other hand, we ignore inputs which result in no recommendations and report $\widetilde{\mathrm{tp}_m}$. Thus, from our earlier introductions of the measures of precision and recall, for individual inputs, we deviate for inputs with no recommendations by ignoring them in the calculations for $\mathrm{pr}_{\mathrm{avg}}$ and $\mathrm{rc}_{\mathrm{avg}}$.

We believe, when considering the choice of a CHB recommender system all five measures

($\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, $\mathrm{tp}'_{\mathrm{avg}}$, $\mathrm{pr}_{\mathrm{avg}}$, and $\mathrm{rc}_{\mathrm{avg}}$) should be taken into account. Appendix A.3 gives the $\widetilde{\mathrm{pr}_m}$, $\widetilde{\mathrm{rc}_m}$, and $\widetilde{\mathrm{tp}_m}$ values for individual modification tasks and $\mathrm{pr}_{\mathrm{avg}}$, $\mathrm{rc}_{\mathrm{avg}}$, and $\mathrm{tp}_{\mathrm{avg}}$ for a range of frequency thresholds (2–20).

3.3 Discussion

We have already discussed some of the differences arising in porting Ying et al.'s approach from file-level to method-level granularity. We now discuss the consequences of a finer granularity on our experimental setup and on results.

3.3.1 Effect of granularity on experimental methodology

We have implemented Ying et al.'s CHB approach at two granularities: fine-grained (at method-level) and coarse-grained (at file-level). When adapting Ying et al.'s original approach to a fine-grained granularity we made the decision to only consider methods. This implies that we ignore other fine-grained artifacts of a Java class like import declarations and fields. While methods form the bulk of the code most of the times for a given class, including fields and import declarations in the fine-grained implementations of both the CHB and ECHB approaches would make our implementation complete in terms of code coverage.

Our decision to ignore import declarations and fields was driven by the nature of our intended extension to the original work in the form of ECHB. Comparing methods for similarity and hence possible transformation is more straight forward as the five facts (name, return type, parameters, callers, and callees) encompass a structure of a method completely. However, to identify with certainty the origins of a transformed field, we would have to consider the usage of the field which is more complex and beyond the scope of this thesis (see Chapter 9).

Difference in granularity had some interesting effects on the experimental setup. For example, for four of the 20 modification tasks under consideration (Bugs #24668, #24449, #24662,

and #24756), the solution set consisted of only a single file (after filtering out the build-notes file). We did not use these tasks for the experiments, in the class level, as Ying et al.'s CHB approach is intended to be used for modification tasks involving *multiple* files. Interestingly, for our method-level implementation, these four tasks could still be used for experiments as the tasks consisted of multiple methods belonging to a single class. Appendix A.5 gives the results for the method-level implementation of CHB approach without these four tasks for a better comparison with the file-level results. The overall effect of removing these tasks is nominal as three of the four tasks did not result in valid recommendations. The individual throughput for these tasks was also small.

3.3.2 Effect of granularity on results

Table 3.1 shows the results for first four thresholds of CHB recommender approach for method-level and file-level granularity.

ω	CHB file-level			CHB method-level		
	$\mathrm{pr}'_{\mathrm{avg}}$	$\mathrm{rc}'_{\mathrm{avg}}$	$\mathrm{tp}'_{\mathrm{avg}}$	$\mathrm{pr}'_{\mathrm{avg}}$	$\mathrm{rc}'_{\mathrm{avg}}$	$\mathrm{tp}'_{\mathrm{avg}}$
2	0.23	0.58	0.87	0.35	0.21	0.24
3	0.38	0.51	0.85	0.34	0.29	0.16
4	0.43	0.44	0.81	0.48	0.32	0.13
5	0.42	0.39	0.78	0.53	0.29	0.11

Table 3.1: Comparison of results for method-level and file-level implementation of CHB approach. ω represents the threshold for the respective granularity.

If we compare the $\mathrm{pr}'_{\mathrm{avg}}$ values for the two approaches we can see that for the same threshold, the method-level CHB approach results in higher $\mathrm{pr}'_{\mathrm{avg}}$ values for 3 out of 4 thresholds. As we can see, the general trend is that, for lower thresholds, method-level CHB approach tends to perform better in terms of $\mathrm{pr}'_{\mathrm{avg}}$. Threshold 3 is an anomaly perhaps because for the increase in threshold from 2 to 3 many inputs do not result in recommendations (as seen in the sharp drop in $\mathrm{tp}'_{\mathrm{avg}}$), but the inputs that do result in valid recommendations contain a lot of incorrect

recommendations.

On the other hand, $\mathrm{rc}'_{\mathrm{avg}}$ and $\mathrm{tp}'_{\mathrm{avg}}$ drop sharply when moving from file-level to method-level results. Thus, in general, the finer the granularity of an artifact in a system, the less likely it is to change. For example, if a class has 5 methods, each is changed once, then the total number of changes to the class is already 5 which is 5 times the number of changes to each method.

The choice of appropriate granularity depends on the project for which the recommender system is being used and the use case scenario. A developer wanting to understand the effects of a high level design decision and its implementation through change tasks might find coarse-grained recommendations more useful. This could be quite useful in planning and estimating the time needed to complete a bug fix. Whereas a developer who is involved in doing the actual modifications, will be more interested in fine-grained recommendations.

For our method-level implementation of Ying et al.'s CHB approach, at times the recommendation and the input belong to the same class. It can be argued that such a recommendation cannot be of much use to developer as it would be an *obvious* suggestion to the developer. Analogously, the same can be argued for the file-level variant of the approach too. A file recommendation belonging to the same package as the input, where the package has few inputs can be already known to the developer. Ying et al. have performed a qualitative analysis in addition to their quantitative experiments. The qualitative analysis groups the recommendations in three categories: *obvious*, *neutral*, and *surprising*. The categories, as their names imply, classify the recommendations in increasing order of usefulness. Their file-level approach was already found to return recommendations that could not be captured by current static and dynamic analysis techniques (belonging to *surprising* category). A finer granularity will only help in producing more results in the *surprising* category, which is the most useful category of recommendations.

As our method-level implementation of their approach is based on the same principles,

the subjective categories defined by Ying et al. will also apply to method-level recommendations. Thus, for example, if `Shape.java` (input) results in `Utility.java` as a *surprising* recommendation, then, `Shape.java/draw()` resulting in `Utility.java/draw()` will also belong to *surprising* category.

Recommendations belonging to the same class as input for method-level granularity might not be always obvious to developers. Several classes in Eclipse contain dozens of methods, making it difficult to narrow down to the methods of interest when working on a modification task. An example of such a class in Eclipse is `org.eclipse.update.internal.core.-UpdateManagerUtils.java` containing more than 40 methods. A recommendation for such large classes, which help trim down the number of relevant methods for the developers can be useful despite belonging to the same class. Even in cases where a recommendation is *obvious* it can still be useful by reinforcing results from static analysis approaches or developer's knowledge of the code base.

Thus, it is evident from various experimental results that CHB recommender approach at both file-level and method-level granularity can provide useful recommendations to developers. However, exactly how a particular granularity can be employed in real world scenario and the variation in usefulness of the results to the developers based on various factors (developer's knowledge of the system, task at hand, nature of the system) can only be assessed by performing user case studies on a large scale.

3.4 Summary

In this chapter we describe our implementation of Ying et al.'s CHB recommender approach for method-level granularity. When porting from file-level to method-level granularity some variations are unavoidable and we discuss them in detail. In Section 3.1 we provide details of our implementation of Ying et al.'s original approach for method-level granularity. We discuss

all the implementation details where we varied from the file-level implementation of their approach and discuss the effects of these variations. Next, in Section 3.2 we elaborate on our evaluation experiment for the method-level variant. We also discuss some of the shortcomings of Ying et al.'s measures of $\mathrm{pr}_{\mathrm{avg}}$ etc. and introduce additional task-wise measures to better understand the efficacy of the recommender systems at individual task-level. Finally, we discuss the effect of granularity on experimental methodology and results in Section 3.2. We believe that both variants of CHB approach are useful to developers based on the modification task at hand.

Chapter 4

AN ALGORITHM FOR DETERMINING METHOD SIMILARITY

In evolving software systems, many developmental activities can result in changes to the name or other properties of a software entity (e.g., a method or class). Such changes, though common, can result in difficulty in determining that the earlier version and the later version of the entity share a common identity. In version control systems like CVS, a transformed entity would in fact appear as a new entity. Consequently, change history based (CHB) recommendation systems, which rely on the association of historical information to a software entity, experience a loss of information for such transformed entities. To overcome this problem, a means for recovering the lost identity, or otherwise borrowing the historical information from another entity that has it, is needed.

To obtain the historical information of a transformed entity, we must identify entities (methods and classes for our work) that are similar to the transformed entity. Then, we can leverage the history of the similar entity as the history of the transformed entity and proceed to find recommendations like in a CHB recommender system. In this chapter, we describe our algorithm for determining the similarity between pairs of methods; we will apply this algorithm in our similarity based (SB) and extended change history based (ECHB) approaches to borrow historical information from a method that is most similar to the one that is missing it.

In Section 4.1 we describe the extraction of relevant method facts for a given version of a system. Section 4.2 describes our algorithm to determine method similarity between the two versions using the extracted method facts. Last, in Section 4.3 we apply our transformation detection algorithm in a stand alone SB recommender approach (method-level) and repeat our

experiments for this approach.

4.1 Method Fact Extraction

A transformation is a set of operations performed on or using p software entities in a given version, resulting in q software entities in the successive version. In an evolving software system several developmental activities can result in a transformation; however, as we are only interested in the history of software entities, we are concerned primarily with the class of transformations that might result in a loss of history, or, the class of transformations whose knowledge would help in associating some historical information to transformed entities. For example, when an entity is split in two in the successive version, historical information for one or both the entities (depending on if one or no entities retain the name of parent entity), is lost. However, if the transformation involves only minor changes to the code of an entity, resulting in simply a changed entity in the subsequent version, the history is preserved.

Refactoring is a common activity in modern software development, which usually involves the meaning-preserving transformation of one or more software entities [21]. Many refactorings, like RENAME METHOD, MOVE CLASS, MOVE METHOD, EXTRACT SUBCLASS, EXTRACT SUPERCLASS, etc., described in the Refactoring Catalogue [17] result in changes that result in loss of history from the perspective of a version control system. The above mentioned refactorings are some of the more commonly used ones [3], indicating that the problem of loss of history will tend to be common.

The creation of code clones (i.e., where a software entity is copied and modified into a new location) is also an industrially common activity [24]. A code clone will initially possess no historical information, despite the fact that it likely shares many of the issues of its ancestor.

In our work, we have adopted a similarity based approach to overcome the loss of historical information from these activities. When a transformation occurs that modifies the name and/or

the location of a software entity, often most of the original structure of the entity is preserved. We leverage this retained similarity between the original and transformed entity to determine the best candidate from which to borrow historical information.

There *can* be transformations which result in severe modification to the structure and behaviour of the original entity. Admittedly, our approach is unlikely to help in such situations. However, an entity that is markedly different from its original entity can gain little from the history of the original entity. Hence, we focus only on situations in which an entity without sufficient historical information bears significant similarity to an entity with sufficient historical information. We have designed our similarity measure to exploit the assumed structural similarity between the original and transformed entities.

To detect method transformations over different versions of a system we use a similarity measure defined by 5 method facts (quite similar to those used by Godfrey and Zou [45]). The method facts we consider are as follows:

Name. The method name preceded by the unqualified name of its container class, e.g., `Shape.foo()`.

Result Type. The fully-qualified result type (i.e., including `void` for methods) as determined through type resolution.

Callers. The set of methods invoking the method under consideration. The members of this set are considered in terms of simple signature preceded by the unqualified name of its container class. For example, `FPTree.insertTree(ItemList, FPTreeNode)`.

Callees. The set of methods invoked by the method under consideration. Analogously to method callers, the members of this set are considered in terms of simple signature preceded by the unqualified name of its container class.

Parameters. The types of the parameters of the method under consideration. The members of

this set are fully-qualified parameter types.

We have refrained from using fully-qualified names for the *Name* fact, because in systems like Eclipse (used for evaluation of our approach) fully-qualified method names tend to be very long with a large part of name common for methods in the same package. For example, the fully-qualified method names for `JavaProject.JavaProject()` and `JavaProjectElementInfo.JavaProjectElementInfo()` are `org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/JavaProject.JavaProject()` and `org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/JavaProjectElementInfo.JavaProjectElementInfo()` respectively. This results in a skewed similarity in method names for methods in the same package.

We use the Abstract Syntax Tree (AST) framework from the Eclipse Java Development Tools (JDT) to extract these five method facts. The extracted method facts for version n are compared against the method facts for version m[1] to obtain name similarity (σ_{name}), return type similarity ($\sigma_{\text{result type}}$), caller similarity ($\sigma_{\text{caller}}$), callee similarity ($\sigma_{\text{callee}}$), and parameter similarity ($\sigma_{\text{parameter}}$).

In general, individual fact similarities are computed by dividing the size of the intersection set of a particular fact for two methods by the size of their union set. An exception to this is name similarity, which is computed by calculating the length of the longest common substring (LCS) between the method names being compared and normalizing it by dividing the LCS with the average length of the method names. Each of the fact similarities lies between 0 and 1. Result type similarity is either 0 or 1.

When extracting individual facts we ignore duplicate elements. Thus the parameter sets of two methods with parameters (`String`, `Integer`, `String`, `Integer`) and (`String`, `Integer`) respectively would be the same, {`String`, `Integer`}. Again, as we are interested in

[1]In our work, version m implies the preceding version. However as the approach can be used for any two versions, in our description, we choose the term version m, over the more specific "version $n-1$".

a similarity measure which only roughly models method pair similarity we are not concerned about the loss of some information. The individual fact similarities between two methods m_1 and m_2 are given by Equations 4.1–4.5.

$$\sigma_{\text{name}}(m_1, m_2) = 2 \cdot \frac{\text{length}(\text{LCS}(m_1, m_2))}{\text{length}(m_1) + \text{length}(m_2)} \tag{4.1}$$

$$\sigma_{\text{parameter}}(m_1, m_2) = \frac{\text{Parameters}(m_1) \cap \text{Parameters}(m_2)}{\text{Parameters}(m_1) \cup \text{Parameters}(m_2)} \tag{4.2}$$

$$\sigma_{\text{caller}}(m_1, m_2) = \frac{\text{Callers}(m_1) \cap \text{Callers}(m_2)}{\text{Callers}(m_1) \cup \text{Callers}(m_2)} \tag{4.3}$$

$$\sigma_{\text{callee}}(m_1, m_2) = \frac{\text{Callees}(m_1) \cap \text{Callees}(m_2)}{\text{Callees}(m_1) \cup \text{Callees}(m_2)} \tag{4.4}$$

$$\sigma_{\text{result type}}(m_1, m_2) = \begin{cases} 1, & \text{if ResultType}(m_1) \equiv \text{ResultType}(m_2) \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

The individual fact similarities are then combined via a weighted linear model (Equation 4.6) to give the total similarity. For every fact i, its fact similarity σ_i is accorded a weight w_i in the computation of total similarity. We use equal weights for the similarity measure in our experiments. We discuss our choice of equal weights and some of the difficulties around selecting a more sophisticated model to weight individual facts in Section 4.2.2.

$$\sigma_{\text{method}}(m_1, m_2) = \frac{\sum w_i \cdot \sigma_i(m_1, m_2)}{\sum w_i} \tag{4.6}$$

The five method facts in our similarity measure can only detect method pairs that have changed structurally. Methods that have been altered in non-structural ways (e.g., changes in method logic) but retain all the five facts, used in the similarity measure, will be accorded

a perfect similarity value (1.0) despite the changes. However, as our goal is to determine if a rough measure of similarity can help in identifying refactorings and similar code, these measures are reasonable for our problem.

The extracted facts of all the methods are stored in an XML file. Each method forms an XML node; name and return type are stored as properties of the method node, while other facts (callers, callees, and parameters) form the children of the method node. Listing 4.1 illustrates a snippet from one of the XML files for Eclipse version 2.1.

We now describe a sample method-pair and the calculation of σ_{method} for equal weights in detail.

Original method:

Name: `PackageExplorerPart.updateTitle(String)`

Result Type: `boolean`

Callers:

`PackagesFrameSource.frameChanged(TreeFrame),`

`PackageExplorerPart.createPartControl(Composite)`

Callees:

`PackagesMessages.getFormattedString(String, String[]),`

`PackageExplorerPart.getToolTipText(Object),`

`ContentViewer.getInput()`

Parameters: `java.lang.String`

Transformed method:

Name: `PackageExplorerPart.updateTitle(String)`

Result Type: `void`

Callers: `PackagesFrameSource.frameChanged(),`

`PackageExplorerPart.createPartControl(Composite)`

Callees: `PackagesMessages.getFormattedString(String, String[])`,
`PackageExplorerPart.getToolTipText(Object)`,
`ContentViewer.getInput()`,
`WorkbenchPart.setTitle(String)`

Parameters = `java.lang.String`

For equal weights, σ_{method} is as follows:

$$\frac{\sigma_{\text{name}} \cdot 1.0 + \sigma_{\text{result type}} \cdot 1.0 + \sigma_{\text{caller}} \cdot 1.0 + \sigma_{\text{callee}} \cdot 1.0 + \sigma_{\text{parameter}} \cdot 1.0}{5.0}$$
$$= \frac{1.0 + 0.0 + 0.33 + 0.75 + 1.0}{5.0}$$
$$= 0.616$$

4.2 Similarity Graph Construction Algorithm

We construct a graph of similarity relationships in the two versions of the source code to be analyzed. The methods in the two versions form the nodes in the graph. Edges are created through our algorithm between similar methods in the two versions and are annotated with method-pair similarity value.

For every method in version j we obtain methods similar to it in version i. We are more interested in finding all the similar methods in version i rather than a single best match for two reasons. First, a method in version j can result from the transformation of several methods in version i, resulting in multiple matches of interest. Second, as we are not interested in the best matches per se but matches modelling a transformation, a variety of similar matches might be useful. For example, a transformation *t*, modifying initially 2 entities to 3 entities, when correctly identified must return 2 similar entities to each of the 3 final entities. Using only a best match would enforce the assumption that a transformation is a change in a *single* entity resulting in a *single* modified entity. However, transformation resulting from several refactorings

```
<?xml version="1.0" encoding="UTF-8"?>
<MethodFacts>
<VersionNode ProjectName="org.eclipse.jdt.core" VersionNum="
   R2_1">

    <MethodName name="SourceReferenceAction.selectionChanged"
                id="org.eclipse.jdt.ui/ui/org/eclipse/jdt/
                   internal/ui/reorg/SourceReferenceAction.
                   java/selectionChanged(IStructuredSelection)
                   "
                returnType="void">
      <callerMethod id="SelectionDispatchAction.
         dispatchSelectionChanged(ISelection)" />
      <callerMethod id="CutSourceReferencesToClipboardAction.
         selectionChanged(IStructuredSelection)" />
      <calleeMethod id="Action.setEnabled(boolean)" />
      <calleeMethod id="SourceReferenceAction.canOperateOn(
         IStructuredSelection)" />
      <parameter id="org.eclipse.jface.viewers.
         IStructuredSelection" />
    </MethodName>

    <MethodName name="ChangeElementTreeViewer.doUpdateItem" id
       ="org.eclipse.jdt.ui/ui refactoring/org/eclipse/jdt/
       internal/ui/refactoring/ChangeElementTreeViewer.java/
     doUpdateItem(Object, Item)" returnType="void">
      <calleeMethod id="TreeItem.setGrayed(boolean)" />
      <calleeMethod id="ChangeElement.getActive()" />
      <calleeMethod id="TreeItem.setChecked(boolean)" />
      <calleeMethod id="TreeViewer.doUpdateItem(Item, Object)"
          />
      <parameter id="java.lang.Object" />
      <parameter id="org.eclipse.swt.widgets.Item" />
    </MethodName>

</VersionNode>
</MethodFacts>
```

Listing 4.1: XML storage format of method facts.

alone involve changes of the type where multiple entities participate in the change resulting in multiple changed entities. In addition, there can be, in theory, other transformations, not recorded in the refactoring cataloged but adhering to our formal definition of transformation. Thus, for our original approach, we threshold our similarity measure to return matches above a ceratin similarity value. However, in Section 6.2.4 we discuss a variant of our original approach that uses only the best match to detect similar entities. We also discuss the implications of using the best-match estimates in this section.

A brute-force approach to determining the most similar methods between two versions would involve comparing every method in version i of a system to all methods in version j of the system. An individual name comparison would have a running time in $O(pq)$, where p and q are the lengths of the compared method names. If we assume that the name lengths are bounded by a constant value, then the running time would be in $O(1)$. Similarly computation time for other fact comparisons would also be $O(1)$ (assuming an upper bound for the set sizes compared). Thus, if the number of methods in version i is m and in version j is n then effectively the computational complexity of the brute-force approach would be $O(mn)$ or roughly $O(n^2)$. While this asymptotic complexity class is reasonable for small- to medium-sized systems, the computation time can grow excessive for large-scale systems with several thousand methods. A brute-force approach could also result in significant storage complexity.

One way to reduce the computational requirements of the brute-force approach is to compare only those methods in version i for which no (or little) history can be associated. In other words, we compare only those methods which do not have a counterpart in the version j. To do so, we employ a two pass approach, embodied in the CREATE-SIMILARITY-GRAPH algorithm. We begin by storing each name fact from each version into a hashtable (H_i for version i and H_j for version j), using Java's standard hashcode implementation for `String`.

CREATE-SIMILARITY-GRAPH($H_i, H_j, \tau_{identity}, \tau_{similarity}$)

```
1  for each m̂_q ∈ H_j
2      do m ← H_i[name[m̂_q]]
3         if m ≠ NIL
4           then s ← σ_method(m, m̂_q)
5                if s > τ_identity
6                  then identity[m̂_q] ← TRUE
7                       CREATE-EDGE(G, m, m̂_q, s)
8         if identity[m̂_q] = FALSE
9           then for each m_p ∈ H_i
10                do s ← σ_method(m_p, m̂_q)
11                   if s > τ_similarity
12                     then CREATE-EDGE(G, m_p, m̂_q, s)
```

First pass. The first pass (lines 2–7) determines methods that have not had identity-altering transformation from version i to version j, i.e., methods whose name and location have remained unchanged between these versions. In Java, both name and location are encompassed by the fully-qualified name of an entity, so to detect un-transformed methods in version j, we compare its fully-qualified name against that of each method in version i by a simple lookup of the name in H_i (line 2). If such a name exists in H_i, the method likely retains its identity. However, a check is made on the total similarity to ensure that the similarity remains sufficiently high (greater than τ_{identity}) for the methods to be considered the same entity (lines 4–7). Regardless, the method in version i is retained for further processing, since it may be involved in other transformations (e.g., cloning).

Second pass. In the second pass (lines 8–12), all the methods in version j whose identity was not established in the first pass are compared with all the methods in version i. For each

method pair being compared, σ_{method} is calculated (line 10). If the σ_{method} value is greater than $\tau_{\text{similarity}}$, an edge is created between the two methods and marked with σ_{method}(lines 11–12).

Although the asymptotic complexity of this algorithm is still $O(n^2)$, we have considerably reduced the size of the sets to be compared, thereby resulting in faster comparison in most practical situations. We provide empirical results in Section 4.2.1

4.2.1 Measurement of speed-up

We measure the speed-up in performance of our CREATE-SIMILARITY-GRAPH algorithm, as compared to a brute force approach comparing every method in version m against all methods in version n. For this we implemented the brute force approach by modifying our CREATE-SIMILARITY-GRAPH algorithm to eliminate the first pass. Thus, methods with no changes to their fully qualified name (methods identical between version m and version n) were not filtered out.

We tested the speed-up on a Dell Precision PWS490 machine with 8.00 GB of RAM and Microsoft Windows XP Operating System. We used releases R2_0 and R2_1 of Eclipse as our two benchmark versions. Method facts for the two versions were extracted using the same machine. Number of unique methods in R2_1 were 67470, while in R_0 were 57184.

We first ran our CREATE-SIMILARITY-GRAPH algorithm with τ_{identity} as 0.75 and $\tau_{\text{similarity}}$ as 0.1 (the minimum value used in our experiments). Pass 1 of our algorithm finished executed in 2.8 seconds. The first pass identified 30291 methods in R2_1 as identical to their counterparts in R2_0. The $identity[\hat{m}_q]$ of these methods in version R2_1 was marked as 1 and they were not used further in pass 2. We tested our algorithm for a subset of methods in R2_1 of size 600. Pass 2 completed execution in 21.03 minutes for 600 methods and thus effectively our CREATE-SIMILARITY-GRAPH algorithm built the Similarity Graph in just over 21 minutes.

We also ran the CREATE-SIMILARITY-GRAPH algorithm without the check for identity by removing line 11 of the algorithm. The algorithm now identified 43920 methods in R2_1

as having an identical equivalent in version R2_0 and for an input of 600 methods completed in 18.71 minutes. Thus, $identity[\hat{m}_q]$ was useful in distinguishing methods that retained their fully qualified name but changed in non-trivial ways structurally versus methods that retained their fully qualified name and did not change structurally (or changed very little).

Last, we ran our brute force variant of CREATE-SIMILARITY-GRAPH. This algorithm completed execution in 48.07 minutes, thus providing over 200% improvement in speed up. Our CREATE-SIMILARITY-GRAPH algorithm filters out approximately 45% of the methods in the first pass, which is the central optimization in our algorithm. Thus, intuitively, the speed-up should be around 100%. However, as the number of methods in the graph is higher for the brute force variant, the number of edges also is significantly higher, leading to a denser graph which adds to space complexity and thus the speed-up is actually higher than expected.

Eclipse is perhaps unusual in terms of the small number of methods that retained their identity (45%) and the speed-up may be even more for other systems in which a larger number of methods preserve their identity over two versions.

4.2.2 Fact weights in the similarity measure

Currently our approach uses equal weights for the similarity measure. Although a more refined similarity measure experimenting with different weights may return better results, we found that testing our approach for varying weights was difficult. A better similarity measure might involve a non-linear combination of varying weights; however, determining such a measure and testing it for efficacy prior to performing the actual experiments is particularly problematic.

A näive approach for finding a better similarity measure could involve using the previously described handcrafted examples to find a similarity measure which best captures the similarity amongst various method pairs. The problem with this approach is that while the handcrafted test cases can be used to test the correctness of the approach, they are not representative of any real world large software system. A system like Eclipse, which we used as our benchmark sys-

tem for all the experiments, involves many developers and hundreds of modules. Handcrafting test cases that can capture the structural properties of the whole of Eclipse and hence aid in finding the best similarity measure is a research goal in its own right and not the focus of thesis. Even a select set of cases from Eclipse code base itself is not guaranteed to be a sufficient representative of the entire Eclipse code base.

Thus, any variations in the similarity measure can be reasonably tested for their effectiveness only with real modification tasks. However, in the scenario that a perfect measure is determined it is hard to argue that the measure would work for another set of modification tasks in Eclipse, let alone in another system. We believe that assigning equal weights to all facts in our similarity measure is reasonable and would lead to a acceptable similarity values. We think that the use of equal weights can balance the effects of individual facts in determining total similarity.

Ultimately, it is important to remember that the goal of our approach is to substitute missing history of a software entity with the history of other similar entities and the most similar entity might not be the best substitute for the history. Thus, we believe that for our work a similarity measure which is not computationally expensive (as we are dealing with large scale systems like Eclipse) and a reasonable representation of similarity should suffice.

4.3 Standalone SB Approach and its Evaluation

We have implemented our algorithm for detecting transformations, CREATE-SIMILARITY-GRAPH, as a stand alone Similarity based (SB) recommender approach. The general principle of our standalone SB approach is that for any input i from version n, we extract all similar methods in version m using CREATE-SIMILARITY-GRAPH. The set of similar methods in the previous version forms the set of recommendations for the given input.

Our rationale for implementing the standalone approach is two-fold. First, we want to

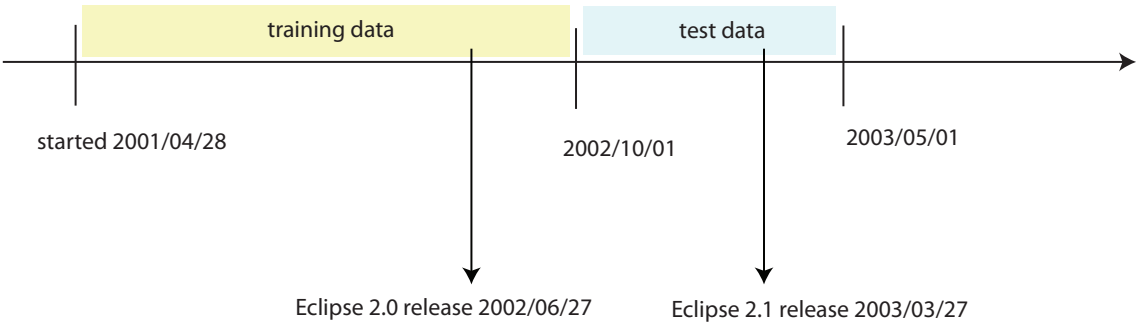

Figure 4.1: Eclipse timeline used for the standalone SB approach.

investigate the role of structurally similar code in bug fixes. Second, since similarity based estimates are the underpinnings of our extended change history based approach (ECHB), we want to investigate the effect of various similarity thresholds on the usefulness of recommendations for our stand alone SB recommender approach. For the evaluation of the similarity based (SB) approach (and later for our ECHB approach) we selected two major releases for Eclipse. It was important to select the releases in a manner that the later release fell in the duration of test data while the earlier release was in the training data. Such a selection of timelines would more likely result in transformed entities in test data, enabling us to see how our SB approach fares with respect to CHB approaches. This selection of timeline allows us to observe the effects of our approach in the context of a major release resulting in transformations. Figure 4.1 depicts the training data and test data timelines along with two major Eclipse releases.

We repeat our experiments as described in the previous chapters, using each entity in the solution set for a bug fix as input. For our similarity based approach we first create a similarity graph using the algorithm CREATE-SIMILARITY-GRAPH and input hashtables for version R2_1 and R2_0 of Eclipse. Next, for each input method in the solution set for the 20 modification tasks used for evaluation, we retrieve the corresponding similar methods from the similarity graph.

For our standalone SB approach, these similar methods form the recommendations. Next, we use the measures $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, $\mathrm{tp}'_{\mathrm{avg}}$ to assess the usefulness of the recommendations. We

vary the similarity threshold from 0.1 to 1.0 (increments of 0.1), however, our approach does not return any results for threshold 1.0.

Figure 4.2 shows the effects of varying similarity threshold on $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$.

For our experiments we obtained the following best values:

1. $\mathrm{pr}'_{\mathrm{avg}} = 1.00$ at similarity threshold 0.9 ($\mathrm{rc}'_{\mathrm{avg}} = 0.03$, $\mathrm{tp}'_{\mathrm{avg}} = 0.001$)

2. $\mathrm{rc}'_{\mathrm{avg}} = 0.20$ at similarity threshold 0.1 ($\mathrm{pr}'_{\mathrm{avg}} = 0.007$, $\mathrm{tp}'_{\mathrm{avg}} = 0.24$)

3. $\mathrm{tp}'_{\mathrm{avg}} = 0.24$ at similarity threshold 0.1 ($\mathrm{pr}'_{\mathrm{avg}} = 0.007$, $\mathrm{rc}'_{\mathrm{avg}} = 0.20$)

As seen in Figure 4.2, the results for the stand alone SB approach are much less promising than the earlier described CHB approaches (for method-level granularity). These low values for the similarity measures are expected. High values for precision and recall would imply that all the entities changed as part of modification tasks are similar to each other, which is seldom the case and reason why CHB approaches are useful. Values for the different measures for our stand alone approach confirm the notion that structural similarity does not capture all the information when modification tasks are resolved.

4.4 Summary

In this chapter we described our algorithm to detect transformations over two versions of a system. We also implement our algorithm through a similarity based (SB) approach to study the usefulness of similarity estimates for making pertinent recommendations. We use method facts to detect method similarity which is an indicator of potential transformations. We first describe the 5 method facts (name, parameters, return type, callers, callees), to detect method-pair similarity in Section 4.1. Next we describe our transformation detection algorithm, CREATE-SIMILARITY-GRAPH, its performance and the difficulties in determining appropriate weights for our similarity measure in Secion 4.2. Finally, we use this algorithm in our stand alone

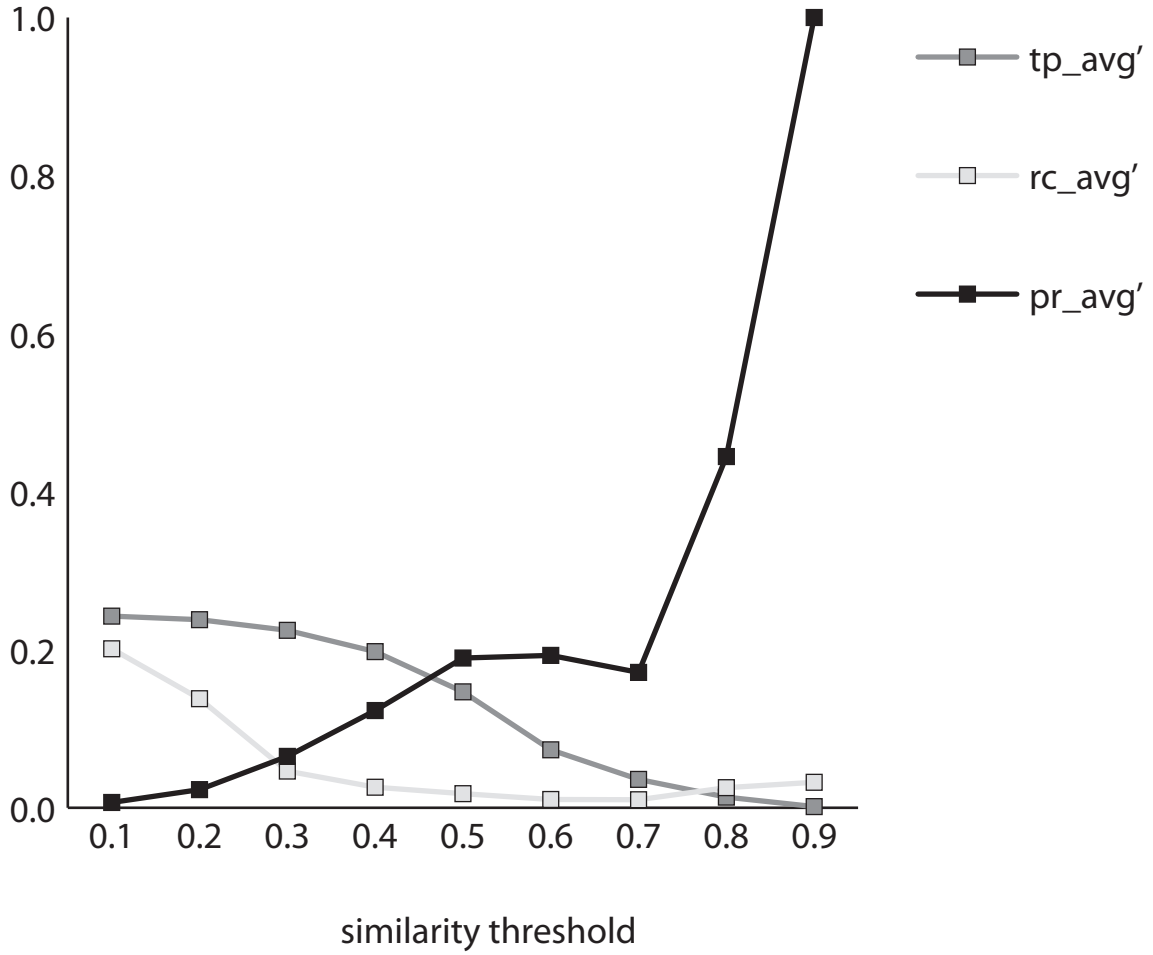

Figure 4.2: Effect of different similarity thresholds on $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$ for the standalone SB approach at method-level granularity.

SB recommender approach and we evaluate the SB recommender approach using the 20 modification tasks from Eclipse. We find that similarity alone is not sufficient to provide useful recommendations, confirming the usefulness of CHB approaches.

Chapter 5

SIMILARITY-BASED APPROACH FOR CLASS-LEVEL GRANULARITY

To compare our standalone SB recommender approach for method-level granularity against class-level granularity, we have extended our method-level approach to the class level. We use method-pair similarities described in 4.1 between methods of two classes to calculate the overall class similarity.

In this chapter, we first describe our technique for computing similarity between class-pairs, in Section 5.1. In Section 5.2 we describe our class-level SB approach based on the class-pair similarity. Last, in Section 5.3, we describe the results of our experiment for the SB recommender approach for class-level granularity.

5.1 Combining Method-Level Similarities to Obtain Class-Level Similarity

Here, we leverage the pre-existing method-similarity calculation technique from Section 4.1 to calculate the overall class-level similarity. Our general technique is as follows:

1. If M_i is the set of methods in class A and M_j is the set of methods in class B, we calculate $\sigma_{\text{method}}(m_i, m_j)$ where[1] $(m_i, m_j) \in M_i \times M_j$.

2. Next, for each $m_i \in M_i$, we find $m \in M_j$ such that $\sigma_{\text{method}}(m_i, m)$ is the highest $\forall m \in M_j$.

3. We repeat Step 2 for each method in class B, that is, for each $m_j \in M_j$, we find $n \in M_i$ such that $\sigma_{\text{method}}(n, m_j)$ is the highest $\forall n \in M_i$.

[1] $M_i \times M_j$ represents the Cartesian cross-product.

4. We add the highest similarity values, obtained through Steps 2 and 3, for best match method-pairs in classes A and B.

5. Last, we normalize the sum of similarity values for best match method-pairs in class A and B by dividing it with the sum of method set sizes of class A and B ($|A| + |B|$).

Intuitively, if any two classes are identical, then each method in the first class would have a corresponding identical method in second class ($\sigma_{\text{method}}(m_i, m_j)$ = 1.0). However, for any specific method in class A, it is highly probable that only one method in class B has $\sigma_{\text{method}}(m_i, m_j)$ = 1.0 and all other methods have lower similarity values.

Thus, by selecting only the method-pairs with highest similarity values, our technique ensures that the non-similar method pairs do not negatively affect the total similarity values.

Equation 5.1 gives the formula for calculating class $\sigma_{\text{class}}(A, B)$.

$$\sigma_{\text{class}}(A, B) = \frac{\sum_{m_i \in A} \max_{m \in B}\{\sigma_{\text{method}}(m_i, m)\} + \sum_{m_j \in B} \max_{n \in A}\{\sigma_{\text{method}}(n, m_j)\}}{|A| + |B|} \tag{5.1}$$

5.2 Class-Level Similarity-Based Approach

We implement our similarity based approach to find similar classes amongst those in versions R2_1 and R2_0 through the following steps.

1. If C_i is the set of classes in version R2_1 and C_j is the set of classes in version R2_0, we calculate $\sigma_{\text{class}}(c_i, c_j)$ where $(c_i, c_j) \in C_i \times C_j$ using the equation 5.1.

2. For a given class $c_i \in C_i$ we create an edge with each the classes $c \in C_j$ with similarity greater than a specified similarity threshold.

Our approach for calculating class similarity for two versions differs from our approach for finding method similarity for two versions described by algorithm CREATE-SIMILARITY-

GRAPH. This is because unlike in CREATE-SIMILARITY-GRAPH we do not filter out identical classes. We cannot filter out any methods because we are using the similarity values for all method-pairs, identical or not. On the other hand, filtering out identical classes is not possible in our current approach as we are building class-similarity from the method-similarity pairs and do not have a way of identifying identical classes easily.

However, to keep the computation time reasonable we do not build the entire similarity graph of the system, but build smaller graphs for each input during our experiment.

5.3 Experiment

In our evaluation, each file in the solution set of a modification task is used as an input. The public Java class in the file is compared against all classes from version R2_0 to obtain a set of classes with similarity above the given threshold. This is done by steps described in Section 5.1. The difference is that in place of constructing the similarity graph for the entire system we create it for a single class input at a time.

Figure 5.1 shows the effects of varying similarity threshold on $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$.

As can be seen in the graph the results from the standalone SB approach for class-level are not strong. The $\mathrm{pr}'_{\mathrm{avg}}$ values are very low and the best value is only about 0.2 at similarity threshold 0.5 for which the $\mathrm{rc}'_{\mathrm{avg}}$ and $\mathrm{tp}'_{\mathrm{avg}}$ values are extremely low (less than 0.01 each). As discussed earlier, in Section 4.3, we suspect that *only* similarity is not a good indicator of relevant classes (or methods) in the context of a modification task.

5.4 Summary

In this chapter we first describe our approach for detecting class similarity using individual method-pair similarities obtained through CREATE-SIMILARITY-GRAPH in Section 5.1. Next, we discuss implementation of our class similarity technique through our SB recommender

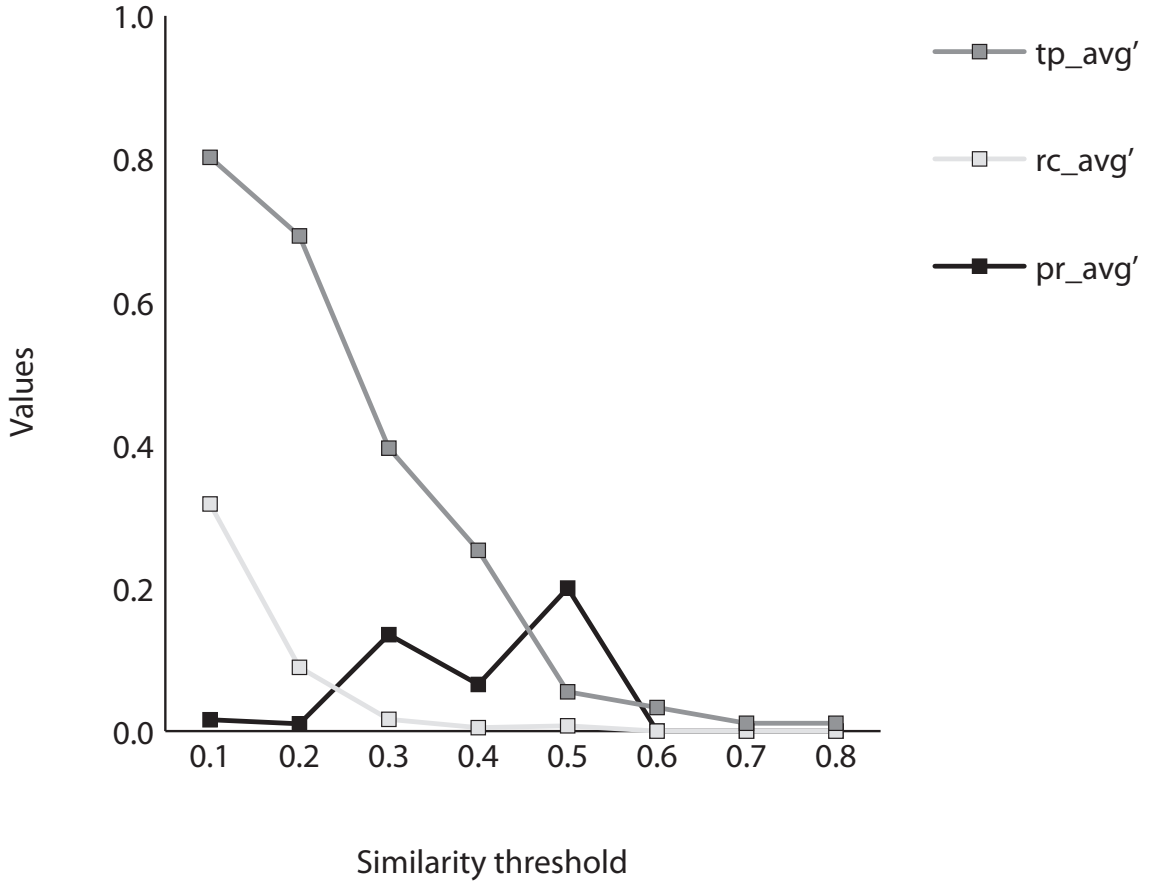

Figure 5.1: Effect of different similarity thresholds on $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$, and $\mathrm{tp}'_{\mathrm{avg}}$ for standalone SB approach at class-level granularity.

approach and its evaluation in sections 5.2 and 5.3 respectively. The results from the SB approach at class level are not strong with very low values for $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$. This further shows that entity similarity alone is not helpful in guiding developers working on modification tasks. However, to assess if similarity estimates (by way of transformations) can enhance CHB approaches, in Chapter 6 we will combine our SB approach at class level with Ying et al.'s original CHB approach at file-level to obtain the extended CHB approach (ECHB).

Chapter 6

COMBINING CHANGE HISTORY BASED AND SIMILARITY BASED ESTIMATES

Current change history based (CHB) recommender systems do not prescribe an alternative for software entities with limited or no history. This can undermine the results of CHB recommender systems when transformed or newly created entities are used as an input. We believe that, for these entities, using a similarity based approach for detecting similar entities and using the similar entities' history as a substitute for missing history can be a viable solution.

In Chapters 2 and 3, we described the file-level and method-level implementations and the evaluation of Ying et al.'s CHB recommender approach. In this chapter we describe our extended change history based (ECHB) approach for providing useful recommendations to developers working on modification tasks.

First, in Section 6.1 we describe our general approach for combining CHB and similarity based (SB) approaches to create the ECHB approach. Next, in Section 6.2 we describe our experiments to test our ECHB approach for different combinations of the CHB and SB approaches. We propose and implement two variants of our ECHB approach: (a) the best-match (BM) variant and (b) the threshold (TH) variant. In Section 6.3, we describe results from the CHB and ECHB approaches for method-level implementation for inputs satisfying a specific condition so as to remove the confounding factors and include only inputs exhibiting characteristics of transformations. In Secion 6.4 we implement and evaluate our ECHB approach for finding similar substitutes to missing recommendations. Finally, in Section 6.5 we talk about some of the issues around combining change history and similarity estimates.

6.1 Extended CHB Approach

The general approach for ECHB is constructed as follows. For entities with little or no history, we detect similar entities in a previous version and use the history associated with these similar entities in place of the history of the original entity to generate recommendations.

Combining the two approaches for a large scale system like Eclipse is expensive in terms of resources and requires considerable time. Hence we combine the two approaches for specific inputs on the fly. The data associated with each of the individual approaches like extracted facts and generated frequent patterns is still obtained beforehand. However, instead of filling in the historical gaps for the entire system we invoke our similarity-based approach only for the inputs that do not result in any recommendations for the original CHB approach.

The following steps summarize our method for combining the change history and similarity estimates:

1. Each method/class in the solution set for a modification task is used as an input to CHB.
2. Results from the CHB approach are stored as a recommendation set.
3. Now, each inputs for which the recommendation set is null is used as input to the similarity algorithm implemented through our SB approach.
4. For these inputs SB approach finds similar entity/entities (based on the variant used, described later). The SB approach is invoked using versions R2_0 (earlier version) and R2_1 (later version) of Eclipse.
5. CHB approach is now repeated for this set of similar entities, each used as an input.
6. The combined recommendations from all the similar entities are treated as the recommendations from the ECHB approach.

6.2 Evaluation

Our CHB and SB approaches can be combined in several ways. In this section we describe two variants of our ECHB approach and present the evaluation results for both. The results are presented for both method-level and file-level granularities. The variability in our two ECHB approaches comes from the way the similarity algorithm is leveraged to fill in the historical gaps.

6.2.1 TH variant: Implementation and evaluation

The first variant of our approach is an implementation of the general approach (see Section 6.1). For all inputs that result in no recommendations, we find similar entities using our SB approach. These similar entities are determined based on a similarity threshold which is an input to CREATE-SIMILARITY-GRAPH. We use the notations σ_{method} for the method-level similarity threshold and σ_{class} for the class-level similarity threshold for succinctness in this chapter.

Recommendations are now obtained from the CHB approach using these similar entities as inputs. The set of recommendations thus obtained from the set of similar entities is treated as the recommendations from the original input when calculating pr'_{avg}, etc.

In Figure 6.1, label a indicates the results for a single input using the CHB approach, when the input results in valid recommendations. Label b_1 shows the execution path for the same input if it results in no recommendations. Next, our SB approach is invoked on the input with no recommendations to obtain entities similar to it (label b_2), which are then used as input to the CHB approach to generate recommendations (label b_3).

We implemented the TH variant of our ECHB approach for method-level and file-level granularity and repeated our experiments for both.

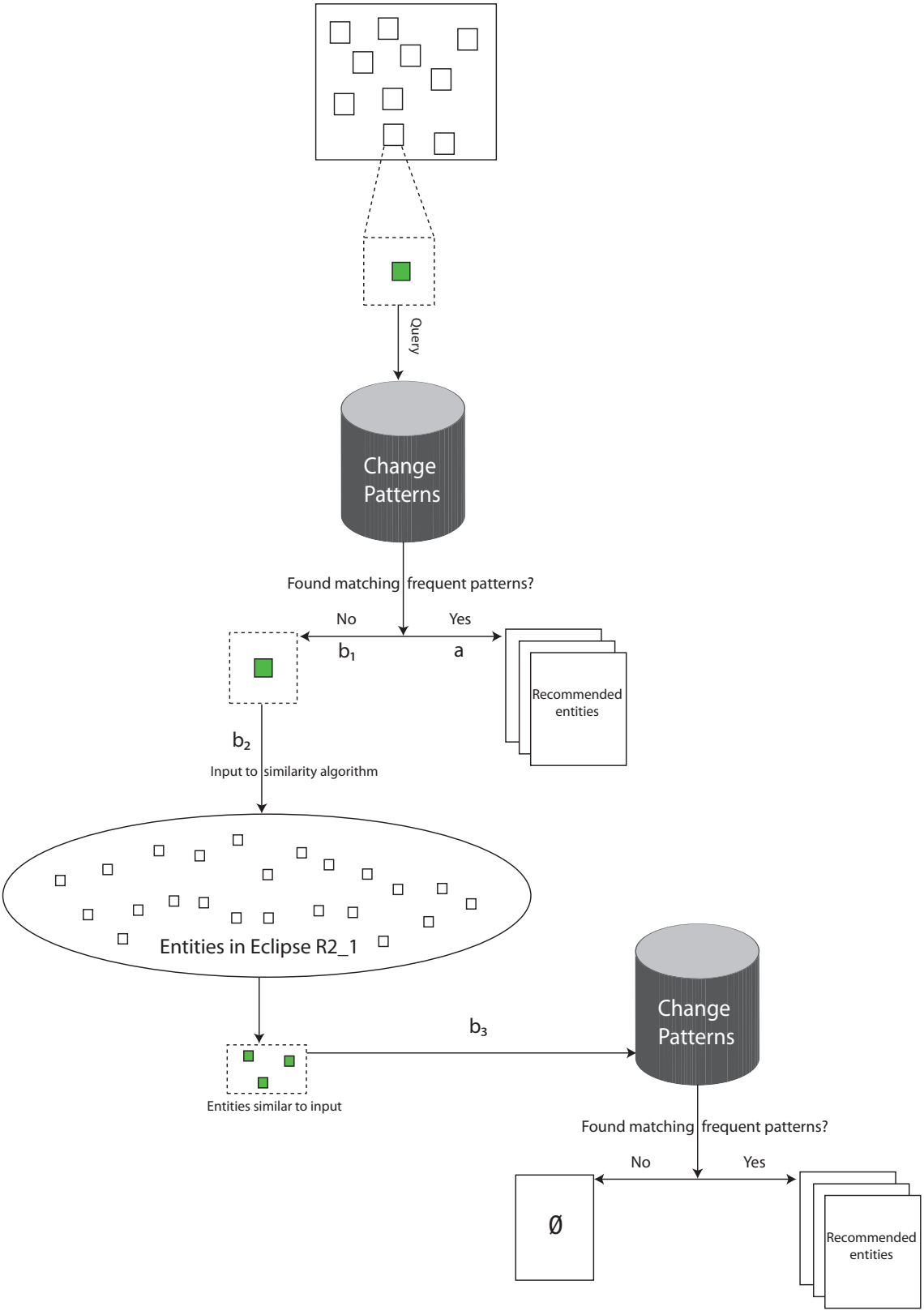

Figure 6.1: Extended change history based approach, TH variant.

6.2.2 Results for ECHB, TH variant, at method-level

For the method-level experiment, we varied the similarity thresholds in the range 1.0 to 0.4 in decrement of 0.1 for the similarity algorithm. We varied the frequency thresholds (ω_{method}) in the range 2 to 21 (decrement of 1) for the CHB approach. We stopped repeating our experiments after a similarity threshold of 0.4 for two reasons:

- The drop in measures of $\mathrm{pr}'_{\text{avg}}$ and $\mathrm{rc}'_{\text{avg}}$ becomes sharper as we decrease the similarity threshold thus diminishing the usefulness of the results with decrease in the threshold.
- Currently, for very low similarity thresholds, for method-level granularity, our tool cannot provide recommendations due to computational resource limitations.

The choice of best threshold for the CHB approach is a difficult one and depends on the use case for which CHB is applied (see Section 3.3 for a discussion on the difficulties in selecting the best threshold). Nonetheless, we selected four thresholds which gave relatively strong values for each of $\mathrm{pr}'_{\text{avg}}$, $\mathrm{rc}'_{\text{avg}}$, and $\mathrm{tp}'_{\text{avg}}$ individually, without compromising much on the remaining two measures. For example, threshold 17 results in the best value for $\mathrm{pr}'_{\text{avg}}$ (0.89) but $\mathrm{rc}'_{\text{avg}}$ and $\mathrm{tp}'_{\text{avg}}$ (0.12 and 0.01 respectively) are extremely low. Table 6.1 gives the $\mathrm{pr}'_{\text{avg}}$ results for four frequency thresholds (2, 3, 4 and 5) and six similarity thresholds (0.4, 0.5, 0.6, 0.7, 0.8 and 0.9). Tables 6.2 and 6.3 gives the comparative results for $\mathrm{rc}'_{\text{avg}}$ values and $\mathrm{tp}'_{\text{avg}}$ values respectively.

ω_{method}	CHB $\mathrm{pr}'_{\text{avg}}$	ECHB $\mathrm{pr}'_{\text{avg}}$ for $\tau_{\sigma_{\text{method}}} =$					
		0.4	0.5	0.6	0.7	0.8	0.9
2	0.35	0.28	0.31	0.32	0.34	0.35	0.35
3	0.34	0.25	0.28	0.30	0.32	0.34	0.34
4	0.48	0.32	0.40	0.44	0.46	0.47	0.48
5	0.53	0.33	0.43	0.48	0.51	0.52	0.53

Table 6.1: ω_{method} versus $\mathrm{pr}'_{\text{avg}}$ for CHB and ECHB, TH variant.

ω_{method}	CHB rc'_{avg}	ECHB rc'_{avg} for $\tau_{\sigma_{\text{method}}} =$					
		0.4	0.5	0.6	0.7	0.8	0.9
2	0.21	0.18	0.19	0.20	0.20	0.21	0.21
3	0.29	0.21	0.23	0.25	0.27	0.28	0.29
4	0.32	0.22	0.26	0.28	0.31	0.31	0.32
5	0.29	0.18	0.23	0.26	0.28	0.29	0.29

Table 6.2: ω_{method} versus rc'_{avg} for CHB and ECHB, TH variant.

ω_{method}	CHB tp'_{avg}	ECHB tp'_{avg} for $\tau_{\sigma_{\text{method}}} =$					
		0.4	0.5	0.6	0.7	0.8	0.9
2	0.24	0.31	0.28	0.26	0.25	0.25	0.24
3	0.16	0.23	0.21	0.19	0.18	0.17	0.16
4	0.13	0.20	0.17	0.15	0.14	0.14	0.13
5	0.11	0.19	0.15	0.12	0.11	0.11	0.11

Table 6.3: ω_{method} versus tp'_{avg} for CHB and ECHB, TH variant.

The results from our first variant are not very strong. For $\tau_{\sigma_{\text{method}}} = 0.4$ (which gives the best values for pr'_{avg} and rc'_{avg} for ECHB), there is an increase in tp'_{avg} for all four frequency thresholds (2, 3, 4, 5). However, we also see a moderate drop in precision pr'_{avg} and rc'_{avg} compared to CHB. As the $\tau_{\sigma_{\text{method}}}$is increased from 0.4 for 0.1 increments the gain in tp'_{avg} compared to CHB starts decreasing and for $\tau_{\sigma_{\text{method}}}$0.9 and 1.0 the values for the three measures are the same as that of CHB.

The results imply that for most cases when CHB does not return any recommendations our TH variant of ECHB approach either does not return any results or returns results with lower values for the three measures compared to the mean values of CHB (lowering the net values for ECHB). For $\tau_{\sigma_{\text{method}}} = 0.4$, the increase in tp'_{avg} is substantial (0.24 to 0.31); however, the drop in pr'_{avg} values is also analogously high.

On further investigation, we noticed that the results from our SB approach when used as an input to the CHB approach resulted in very large number of recommendations. To elaborate, the similar entities were obtained by varying the similarity threshold from 1.0 to 0.4. As we

decreased the similarity threshold, the set sizes of similar entities for a given input increased drastically. This increase is expected, since, for a large system like Eclipse a similarity threshold of 0.1, for example, would return several hundred methods as similar. Thus, if we assume that the number of results from the CHB approach are a constant k_1 (for a specific frequency threshold) and the number of results from the SB approach are a constant k_2 (for a specific similarity threshold), then, for each input that did not return any recommendations for the CHB approach, the number of recommendations from the ECHB approach for this input would be $k_1 \cdot k_2$, where k_2 is very large for lower similarity thresholds.

For example, the column "# Similar entities" in Table 6.4 shows the increase in number of similar entities as the similarity threshold is decreased for input `org.eclipse.update.ui/src/org/eclipse/update/internal/ui/forms/DetailsForm/doButtonSelected()`. The column "# Recommendations CHB" shows the increase in number of recommended methods as the frequency threshold is decreased.

σ_{method}	# Similar entities	ω_{method}	# CHB recommendations
1.0	0	20	0
0.9	0	18	0
0.8	0	16	0
0.7	0	14	0
0.6	0	12	0
0.5	0	10	0
0.4	0	8	0
0.3	30	6	0
0.2	27741	4	7
0.1	27748	2	26

Table 6.4: Variation in recommendation set sizes with decreasing similarity and frequency thresholds.

To counter the problem of a large number of recommendations, leading to lowered values of $\mathrm{pr}'_{\text{avg}}$ and $\mathrm{rc}'_{\text{avg}}$ we implemented our ECHB approach through BM Variation.

6.2.3 Results for ECHB, TH variant, at file-level

The results for the file-level implementation of the TH variant of ECHB approach are presented in Tables 6.5–6.7. Table 6.5 gives the $\mathrm{pr}'_{\mathrm{avg}}$ for $\omega_{\mathrm{file}} = 2, 4, 8 and 12$ and $\tau_{\sigma_{\mathrm{class}}} = 0.1, 0.3, 0.5, 0.7 and 0.9$.

The results for the file-level implementation of our approach are even weaker than our method-level implementation. We suspect that part of the reason is that extending our method-level similarity approach to class-level is tricky. Classes have additional artifacts like fields which we ignore in our similarity measure calculation for simplicity. In addition, perhaps our current similarity measure for class similarity does not model transformations accurately. For example, if a single method is moved from one class to another, our current measure would recognize the similar method but penalize the two classes for the dissimilarity of the remaining methods. We believe that while our approach shows promise for method-level transformation detection, for class-level, further research in the nature of transformations is needed to arrive at a more accurate similarity measure.

ω_{file}	CHB $\mathrm{pr}'_{\mathrm{avg}}$	ECHB $\mathrm{pr}'_{\mathrm{avg}}$ for $\tau_{\sigma_{\mathrm{class}}} =$				
		0.1	0.3	0.5	0.7	0.9
2	0.23	0.22	0.22	0.23	0.23	0.23
3	0.38	0.35	0.36	0.37	0.37	0.38
4	0.43	0.40	0.41	0.42	0.42	0.43
5	0.42	0.38	0.41	0.42	0.42	0.42

Table 6.5: ω_{file} versus $\mathrm{pr}'_{\mathrm{avg}}$ for CHB and ECHB, TH variant.

6.2.4 BM variant: Implementation and evaluation

In this variant, when combining CHB and SB approaches, we use the similar entity which is the best match for a given input as opposed to using all similar entities above a given threshold. This results in roughly the same number of recommendations for ECHB as that of CHB (assuming that for a given threshold CHB provides constant number of recommendations k_1).

ω_{file}	CHB rc'_{avg}	ECHB rc'_{avg} for $\tau_{\sigma_{\text{class}}} =$				
		0.1	0.3	0.5	0.7	0.9
2	0.58	0.58	0.55	0.57	0.57	0.58
3	0.36	0.51	0.49	0.50	0.51	0.51
4	0.44	0.44	0.42	0.43	0.43	0.44
5	0.36	0.39	0.38	0.39	0.39	0.39

Table 6.6: ω_{file} versus rc'_{avg} for CHB and ECHB, TH variant.

ω_{file}	CHB tp'_{avg}	ECHB tp'_{avg} for $\tau_{\sigma_{\text{method}}}$=				
		0.1	0.3	0.5	0.7	0.9
2	0.87	0.93	0.92	0.89	0.88	0.87
3	0.85	0.91	0.90	0.87	0.86	0.85
4	0.81	0.88	0.85	0.82	0.82	0.81
5	0.78	0.87	0.81	0.78	0.78	0.78

Table 6.7: ω_{file} versus tp'_{avg} for CHB and ECHB, TH variant.

The similarity algorithm is modified to return only the best-matched entity. If the best match is not unique, we arbitrarily select only the match that was detected first.

Figure 6.2 shows the execution path for our modified ECHB (BM variant). Label b_2 shows the deviation from our general ECHB (implemented through TH variant) approach.

6.2.5 Results for ECHB, BM variant, at method-level

Table 6.8 gives the results for four frequency thresholds for the BM variant of our ECHB approach for method-level granularity.

ω_{method}	CHB			ECHB		
	pr'_{avg}	rc'_{avg}	tp'_{avg}	pr'_{avg}	rc'_{avg}	tp'_{avg}
2	0.35	0.21	0.24	0.32	0.20	0.28
3	0.34	0.29	0.16	0.30	0.25	0.19
4	0.48	0.32	0.13	0.43	0.29	0.16
5	0.53	0.29	0.11	0.47	0.27	0.13

Table 6.8: ω_{file} versus pr'_{avg}, rc'_{avg}, and tp'_{avg} for CHB and ECHB, BM variant.

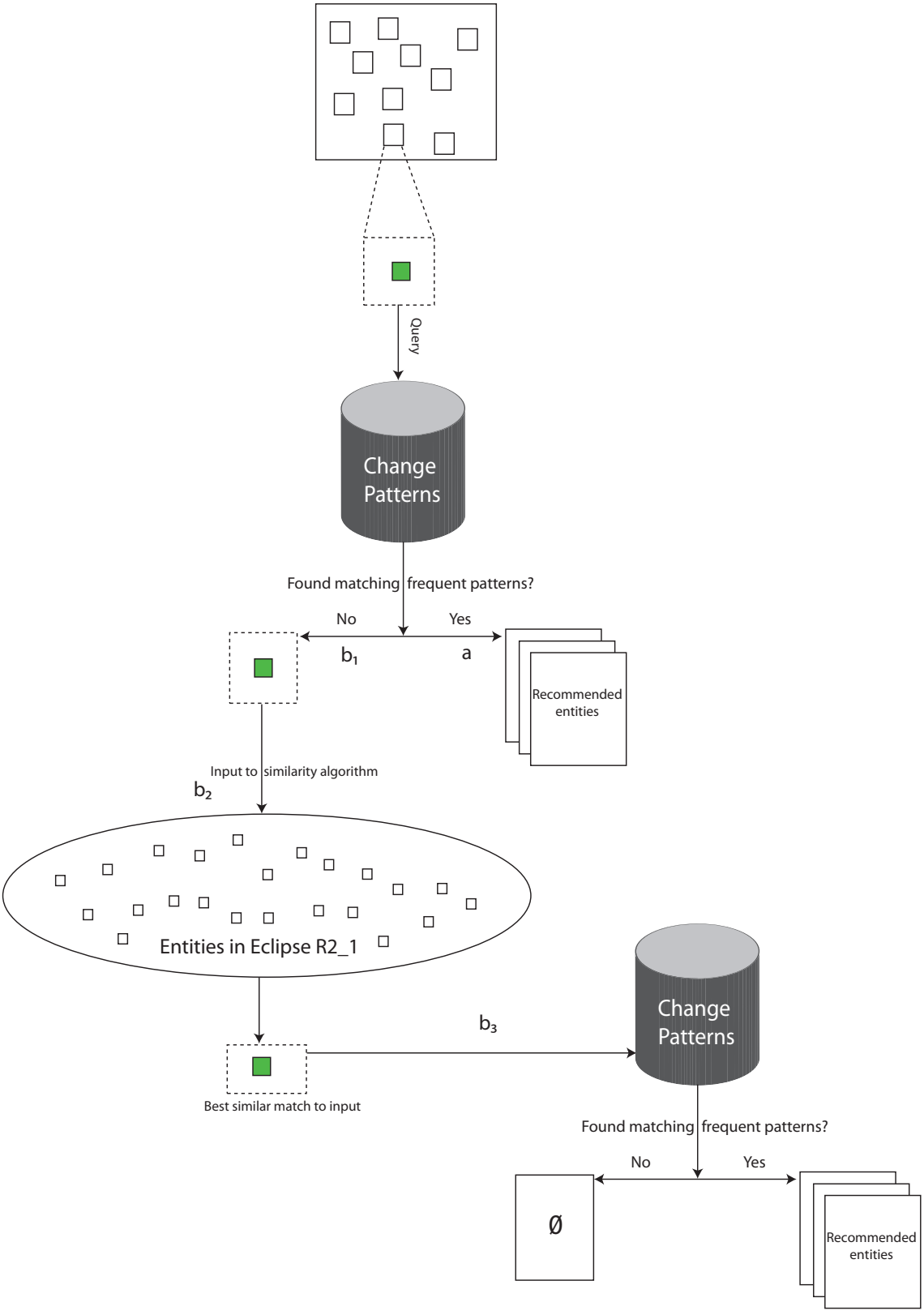

Figure 6.2: Extended change history based approach, BM variant

[BR: Table 6.9 gives the precision and recall values for the individual inputs for the ECHB approach, BM variant at method-level. The values are for the specific cases where CHB approach did not provide any results while the ECHB approach did. NOTE THAT THE REFERENCED TABLE IS ADDED AS PART OF THESIS CORRECTIONS.]

Comparing the results from the BM variant with those from the TH variant, we can observe an improvement in the results. We think this improvement is due to reduction in the excessive number of recommendations. While there still is a drop in $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ values for the ECHB approach as compared to CHB, the drop is smaller. When compared to analogous values of $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ in the TH variant we see that the gain in $\mathrm{tp}'_{\mathrm{avg}}$ is better. For example, for $\omega_{\mathrm{method}} = 2$, BM variant results in $\mathrm{pr}'_{\mathrm{avg}} = 0.32$, $\mathrm{rc}'_{\mathrm{avg}} = 0.20$ and $\mathrm{tp}'_{\mathrm{avg}} = 0.28$. The TH variant, on the other hand, results in $\mathrm{pr}'_{\mathrm{avg}} = 0.32$, $\mathrm{rc}'_{\mathrm{avg}} = 0.20$ and $\mathrm{tp}'_{\mathrm{avg}} = 0.26$ for $\tau_{\sigma_{\mathrm{method}}} = 0.6$.

6.2.6 Results for ECHB, BM variant, at file-level

Table 6.10 gives the results for four frequency thresholds for the BM variant of our ECHB approach for file-level.

We find the the results of the BM variant were only marginally better than TH variant for file-level. Perhaps one of the reasons for weak results is our similarity measure which does not extend well from method-level to class-level. To be able to better analyze the results from method-level ECHB approach in the limited time we do not perform any more evaluation of the ECHB approach for file-level.

6.3 Detailed Analysis of ECHB Results

We realize that our method of comparing CHB and ECHB approaches does not emphasize the specific inputs at which our ECHB approach is aimed at. Our evaluation approach also does not highlight how CHB fares for these cases.

To better understand the scenario for which the ECHB approach is designed, we now explicitly define the nature of inputs for which ECHB approach would be helpful.

Essentially, the ECHB approach should be useful for cases where an input exists in the current version but not in the previous version. For our evaluation experiments, this condition implies that any input that exists in R2_1 but not in R2_0 would not result in any recommendations. This is because for the CHB approach such entities are new and have no history associated with them. However, if this entity is transformed from version R2_0 to R2_1 then our ECHB approach may detect the transformation, substituting the transformed entity's history with that of parent entity.

Thus, in our next experiment we select only those inputs that exist in R2_1 and R2_0 and apply CHB and ECHB approaches to them. Due to the way training data and test data timelines coincide with the release dates of R2_1 and not R2_0 it is possible that CHB approach also gives valid recommendations for some of the inputs. Figure 6.3 shows an entity for which CHB approach may provide valid recommendations even though it satisfies the conditions of a possible transformation. The entity (depicted by "X") is created after R2_0 but still falls within the training data timeline. This is one of the shortcomings of our current evaluation setup. In Section 7.2 we discuss some of the other issues with our experimental setup.

Table 6.11 shows the results from our conditional experiment. We found that out of all the inputs for the method-level CHB approach 72 inputs satisfied the condition of a possible transformation (that is the entity corresponding to the input existed in version R2_1 but not in R2_0). We repeated the method-level CHB and ECHB evaluation experiments for only these inputs. We found that for ω_{method}= 2 almost 50% of the inputs resulted in valid recommendations. While the CHB approach resulted in valid recommendations for only 14% of the inputs. However, the $\mathrm{pr}'_{\mathrm{avg}}$ value was much higher for the CHB approach compared to ECHB approach, $\mathrm{rc}'_{\mathrm{avg}}$ values also were marginally higher for CHB approach compared to ECHB approach.

The results from our conditional experiment are important because they show how our

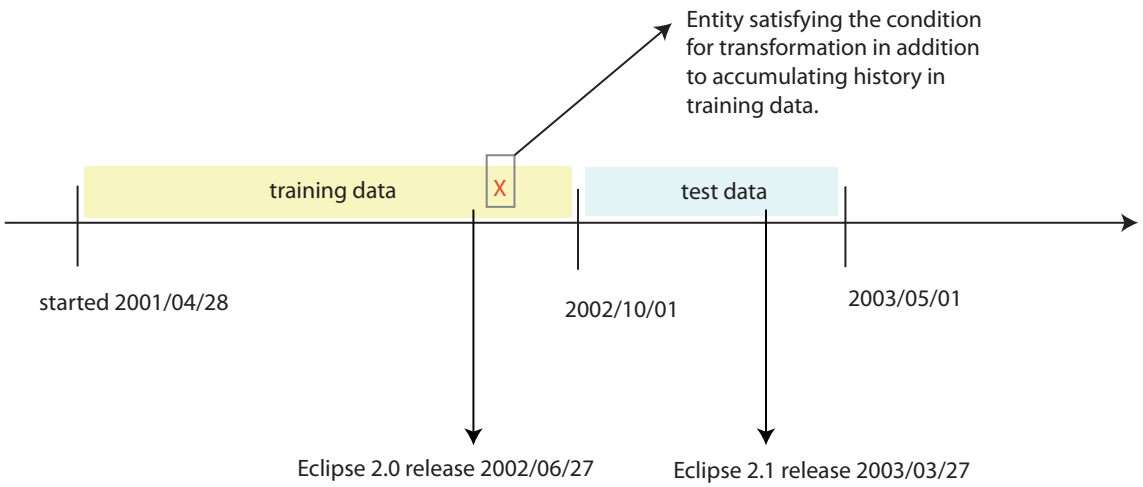

Figure 6.3: Entity satisfying the transformation condition, in addition to having relevant history.

ECHB approach performs for the specific cases for which it is intended. Our results show that ECHB approach provided valid recommendations in about half of the cases where the entity has "disappeared" from the previous version. **[BR: The low $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ values also show that ECHB results are not as strong as that of CHB in general. However, averaging the individual precision and recall values obfuscates a lot of information about the two approaches. To study in detail the cases in which ECHB fares better and vice versa, we present the precision and recall values for each input for the two approaches in tables 6.12(a)-6.12(b). For brevity we present the results for only $\omega_{\mathbf{method}}$= 2.]**

[BR: In tables 6.12(a)-6.12(b), column "Bug ID" gives the identifier of the bug, column "Input" gives the method input (only container class name and method name for brevity), columns "p" and "r" gives the precision and recall values for ECHB and CHB approaches. An entry "N/A" represents the cases where one of the two approaches did not return any results and hence there are no precision and recall values. The table shows the inputs where for at least one of the two approaches, CHB and ECHB, valid results are obtained. For most cases CHB approach does not provide any results, however, for the cases where both approaches provide recommendations, precision and recall values for CHB

approach are better compared to those of ECHB. NOTE THAT THE REFERENCED TABLE IS ADDED AS PART OF THESIS CORRECTIONS.]

A future direction in evaluating ECHB could be to use an additional measure to differentiate inputs of interest for ECHB by using only those entities that have no history associated with them, in lieu of our conditional measure. This additional condition can further help in applying ECHB approach for the scenario it is designed for.

6.4 Using Similarity Estimates to Substitute Non-existent Recommended Entities

Similarity estimates can be helpful not just for substituting history for transformed entities but also for transformed recommendations. For example, if the recommendation provided for a given input entity has transformed since the training data time period, the recommendation would be actually for a non-existent entity. If we can detect the transformation, the transformed entity can be replaced by its currently missing original counterpart which was recommended. If a significant number of recommendations have transformed, we hypothesize that our transformation detection technique can be helpful in the reverse direction too.

Figure 6.4 shows the implementation of the BM variant of our ECHB approach for non-existent recommendations.

Table 6.13 shows the results from our experiments when ECHB approach is applied in both directions, for transformed input entities and transformed recommendations. Surprisingly, the results remain unchanged from that of ECHB, BM variant.

The reason why we do not see any change in the quality of recommendations is because in this bi-directional a set of non-existent recommendations was replaced by a set of incorrect recommendations. Thus we do not see any changes in results. However, it is still worth while to convert our ECHB approach to bi-directional, since, compared to its uni-directional variant

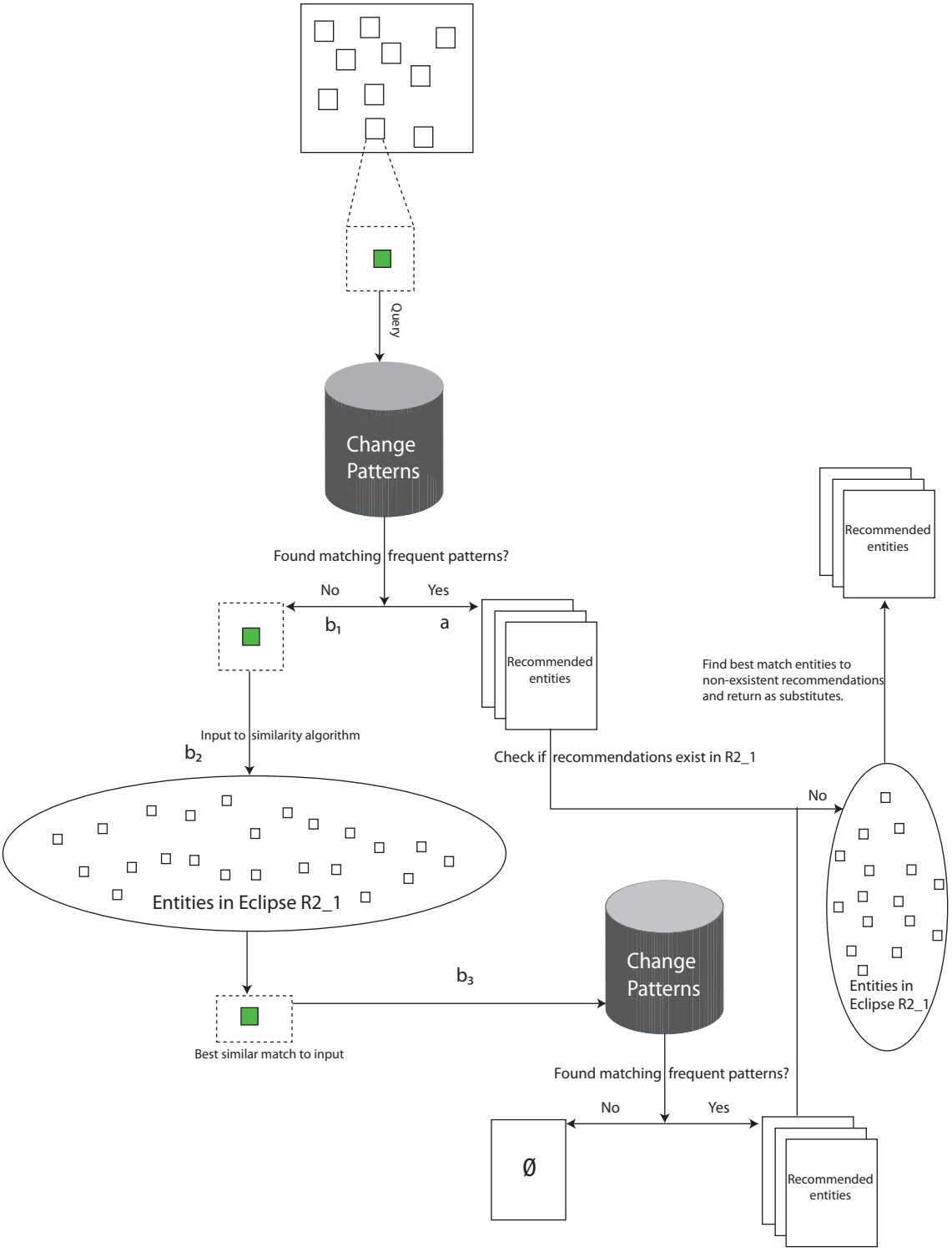

Figure 6.4: Extended change history based approach, substituting non-existent recommendations.

the results would always remain unchanged or improve.

6.5 Discussion

6.5.1 Effect of granularity on ECHB approach

Our ECHB approach is not as effective when ported to class-level granularity. Both variants result in little or no improvement over its CHB counterpart. We think the main reason is differences in our similarity detection technique for class-level compared to method-level. Defining and identifying similarity for method-pairs is simpler than detecting similarity for class-pairs. Our technique for detecting method-pair similarity does not scale well for detecting class-pair similarity. There can be multiple reasons. Our technique utilizes the structural properties for a method (method facts in this thesis), however, for class-pair similarity we combine the results from the method-pair similarity. Perhaps the lost information, like fields in the class, is crucial in determining class-pair similarity. It is also possible that the manner in which we combine method-pair similarity to obtain class-pair similarity is not effective.We feel that further research in class-similarity is necessary before our ECHB approach can provide useful results at this granularity.

6.5.2 Efficacy of ECHB approach compared to CHB

We believe that the strongest results showing the potential of ECHB approach are the ones where we use only those inputs which satisfy the conditions for transformations (see Section 6.3). These results show that when conditions of a transformation are satisfied our ECHB approach can provide relevant recommendations in higher number of cases than CHB (49% as compared to 14%) for method-level granularity. In fact we believe that in a real scenario the number of cases where CHB would provide recommendations in case of a transformation would be even lower. In figure 6.3 we show the limitations of our experimental setup and why

CHB approach provides recommendations in 14% of the cases in our evaluation experiment. For class-level granularity the results for our ECHB approach were not strong (Section 6.2.3 and 6.2.6 gives the results for the two variants). After analyzing the weak results from the ECHB results for file-level granularity we do not pursue further analysis of ECHB approach at file-level by performing the conditional experiment and bi-directional experiment. Similarly for method-level granularity we perform the conditional experiment and bi-directional experiments only for the BM variant, since, the results from the BM variant were stronger than for the TH variant.

6.6 Summary

In this chapter we extend Ying et al.'s CHB approach by filling in the historical gaps using similarity estimates from our SB approach. In Section 6.1 we introduce our general ECHB approach. The idea behind our ECHB approach is that the inputs for which CHB approach fails to provide any recommendations are deemed as transformed. Thus, for these inputs we apply our SB approach to identity the "original" entity/entities. We now invoke CHB approach for the set of "original" entities. Thus, we aim to detect and leverage transformation information to aid CHB approach through our ECHB approach.

Next, in Section 6.2 we implement and evaluate two variants of our ECHB approach for both granularities. The TH variant uses all entities above a certain threshold as a substitute for inputs that do not result in any recommendations. Whereas, in the BM variant, we substitute the input without recommendations with its best-match. The results from these two variants were mixed, while the results for file-level granularity were weak for both the variants, for method-level BM variant, we saw a small improvement in $\mathrm{tp}'_{\mathrm{avg}}$ with a relatively smaller drop in $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ values.

We realized that in our original experiments it is difficult to distinguish the exact inputs

where the cause of no recommendations is transformations. Thus to better study the efficacy of our approach for *only* transformed entities, we performed a conditional experiment where we employ only those inputs that satisfy the "transformation condition", i.e. inputs that exist in version R2_1 but not in R2_0 of Eclipse. The results showed that our ECHB approach provided valid recommendations in 49% cases, whereas, the CHB approach provided recommendations in only 14% cases. However, the $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ values for CHB approach in these 14% cases was higher than ECHB approach (CHB approach: $\mathrm{pr}'_{\mathrm{avg}}$ = 0.47 and $\mathrm{rc}'_{\mathrm{avg}}$ 0.18; ECHB approach: $\mathrm{pr}'_{\mathrm{avg}}$ = 0.09 and $\mathrm{rc}'_{\mathrm{avg}}$ = 0.13[1]).

In Secion 6.4 we implement and evaluate our ECHB approach for finding similar substitutes to missing recommendations.

Finally, in Section 6.5 we discuss the effect of granularity on ECHB approach and the efficacy of ECHB approach over CHB. To conclude we feel that our results show that for method-level variant our ECHB approach shows promising results, however, additional research is needed before we can establish the extent to which ECHB can be an improvement over CHB approaches.

[1]Note that for brevity we compare only results for frequency threshold 2 here.

Bug ID	Input	ECHB	
		p	r
21330	`DetailsForm/executeOptionalInstall`	0.33	0.11
21330	`InstallWizard/ preserveOriginatingURLs`	0.03	0.06
24657	`FeatureReference/FeatureReference`	0.00	0.00
24657	`Utilities/createLocalFile`	0.20	0.03
24657	`UpdateManagerLogWriter/getAction`	0.00	0.00
25041	`PackageFragment/openWhenClosed`	0.02	0.05
25041	`ClassFile/openWhenClosed`	0.02	0.05
23096	`ASTConverter/ASTConverter`	0.02	0.20
23587	`SimilarElementsRequestor/ findSimilarElement`	0.43	0.09
23587	`ASTResolving/getPossibleReference Binding`	0.30	0.09
23587	`NewVariableCompletionProposal/ NewVariableCompletionProposal`	0.09	0.03
23587	`NewCUCompletionUsingWizardProposal/ getAdditionalProposalInfo`	0.01	0.03
23587	`ASTResolving/getBindingOfParentType`	0.17	0.06
23587	`NewCUCompletionUsingWizardProposal/ fillInWizardPageName`	0.01	0.03
23587	`UnresolvedElementsSubProcessor/ createTypeRefChangeProposal`	0.13	0.18
24730	`CVSRepositoryLocation/getRemoteFile`	0.03	0.33
24730	`ICVSRepositoryLocation/ getRemoteFile`	0.08	0.33
16817	`Breakpoint/markerExists`	0.00	0.00
24594	`InternalAntRunner/processTargets`	0.05	0.33
24594	`InternalAntRunner/getArgument`	0.05	0.33
24828	`AppletMainTab/createControl`	0.50	0.04
24828	`JavaAppletLaunchShortcut/ createConfiguration`	0.00	0.00
24828	`AppletMainTab/initializeFrom`	0.00	0.00
24828	`AppletMainTab/ handleSearchButtonSelected`	0.00	0.00
25133	`InternalAntRunner/addInputHandler`	0.08	0.17
25133	`InternalAntRunner/createLogFile`	0.06	0.11
25133	`InternalAntRunner/loadPropertyFiles`	0.08	0.17
Average		**0.10**	**0.10**

Table 6.9: Precision and recall values for individual inputs for ECHB, BM variant.

ω_{file}	CHB			ECHB		
	pr'_{avg}	rc'_{avg}	tp'_{avg}	pr'_{avg}	rc'_{avg}	tp'_{avg}
2	0.23	0.58	0.87	0.22	0.56	0.90
3	0.38	0.51	0.85	0.37	0.50	0.88
4	0.43	0.44	0.81	0.42	0.43	0.82
5	0.42	0.39	0.78	0.42	0.39	0.78

Table 6.10: ω_{file} versus pr'_{avg}, rc'_{avg}, and tp'_{avg} for CHB and ECHB, BM variant.

ω_{method}	CHB			ECHB		
	pr'_{avg}	rc'_{avg}	tp'_{avg}	pr'_{avg}	rc'_{avg}	tp'_{avg}
2	0.47	0.18	0.14	0.09	0.13	0.49
3	0.52	0.27	0.07	0.09	0.12	0.36
4	0.33	0.35	0.04	0.10	0.15	0.26
5	0.50	0.03	0.03	0.11	0.17	0.21

Table 6.11: ω_{method} versus pr'_{avg}, rc'_{avg}, and tp'_{avg} for CHB and ECHB, BM variant; limited to entities that exist in only later version.

Bug ID	Input	ECHB		CHB	
		p	r	p	r
21330	`InstallWizard/execute`	0.02	0.06	0.00	0.00
21330	`InstallWizard/ preserveOriginatingURLs`	0.03	0.06	N/A	N/A
21330	`DetailsForm/executeOptionalInstall`	0.33	0.11	N/A	N/A
24657	`FeatureReference/FeatureReference`	0.00	0.00	N/A	N/A
24657	`FeatureContentProvider/ validatePermissions`	0.00	0.00	1.00	0.03
24657	`Utilities/createLocalFile`	0.20	0.03	N/A	N/A
24657	`UpdateManagerLogWriter/getAction`	0.00	0.00	N/A	N/A
24657	`FeatureReference/getName`	N/A	N/A	1.00	0.03
24657	`FeatureContentProvider/ getPermissions`	N/A	N/A	0.20	0.03
25041	`ClassFile/openWhenClosed`	0.02	0.05	N/A	N/A
25041	`PackageFragment/openWhenClosed`	0.02	0.05	N/A	N/A
23096	`ASTConverter/ASTConverter`	0.02	0.20	N/A	N/A
23587	`NewCUCompletionUsingWizardProposal/ getAdditionalProposalInfo`	0.01	0.03	N/A	N/A
23587	`SimilarElementsRequestor/ findSimilarElement`	0.43	0.09	N/A	N/A
23587	`NewCUCompletionUsingWizardProposal/ fillInWizardPageName`	0.01	0.03	N/A	N/A
23587	`NewMethodCompletionProposal/ getRewrite`	0.12	0.15	0.26	0.15
23587	`ASTResolving/getPossibleReference Binding`	0.3	0.09	N/A	N/A
23587	`UnresolvedElementsSubProcessor/ createTypeRefChangeProposal`	0.13	0.18	N/A	N/A
23587	`NewVariableCompletionProposal/ getRewrite`	0.14	0.03	0.57	0.12
23587	`ASTResolving/getBindingOfParentType`	0.17	0.06	N/A	N/A
23587	`NewVariableCompletionProposal/ NewVariableCompletionProposal`	0.09	0.03	N/A	N/A

Table 6.12(a): Precision and recall values for individual inputs for CHB and ECHB, BM variant; limited to entities that exist in only later version, Part 1.

Bug ID	Input	ECHB		CHB	
		p	r	p	r
24730	`CVSRepositoryLocation/getRemoteFile`	0.03	0.33	N/A	N/A
24730	`ICVSRepositoryLocation/ getRemoteFile`	0.08	0.33	N/A	N/A
16817	`Breakpoint/markerExists`	0.00	0.00	N/A	N/A
24594	`InternalAntRunner/ preprocessCommandLine`	0.05	0.33	0.50	0.17
24594	`InternalAntRunner/processTargets`	0.05	0.33	N/A	N/A
24594	`InternalAntRunner/getArgument`	0.05	0.33	N/A	N/A
24756	`InternalAntRunner/run`	0.05	1.00	0.08	1.00
24828	`AppletMainTab/createControl`	0.50	0.04	N/A	N/A
24828	`JavaAppletLaunchShortcut/ createConfiguration`	0.00	0.00	N/A	N/A
24828	`AppletMainTab/initializeFrom`	0.00	0.00	N/A	N/A
24828	`AppletMainTab/ handleSearchButtonSelected`	0.00	0.00	N/A	N/A
25133	`InternalAntRunner/createLogFile`	0.06	0.11	N/A	N/A
25133	`InternalAntRunner/run`	0.05	0.11	0.12	0.17
25133	`InternalAntRunner/loadPropertyFiles`	0.08	0.17	N/A	N/A
25133	`InternalAntRunner/ preprocessCommandLine`	0.07	0.17	1.00	0.11
25133	`InternalAntRunner/addInputHandler`	0.08	0.17	N/A	N/A
Average		**0.09**	**0.13**	**0.47**	**0.18**

Table 6.12(b): Precision and recall values for individual inputs for CHB and ECHB, BM variant; limited to entities that exist in only later version, Part 2.

ω_{method}	ECHB (uni)			ECHB (bi)		
	pr'_{avg}	rc'_{avg}	tp'_{avg}	pr'_{avg}	rc'_{avg}	tp'_{avg}
2	0.32	0.20	0.28	0.32	0.20	0.28
3	0.30	0.25	0.19	0.30	0.25	0.19
4	0.43	0.29	0.16	0.43	0.29	0.16
5	0.47	0.27	0.13	0.47	0.27	0.13

Table 6.13: ω_{method} versus pr'_{avg}, rc'_{avg}, and tp'_{avg} for ECHB, BM uni-directional, and ECHB, BM bi-directional.

Chapter 7

DISCUSSION

In this chapter, we discuss several issues around our work of comparing and combining change history based (CHB) and similarity based (SB) recommender systems and their effects on the presented results. We summarize various possible sources of errors and our efforts in countering them. We also explain some of our more unexpected results in greater detail.

7.1 Selection of Ying et al.'s Approach

Our primary goal for this thesis was to evaluate the efficacy of using similarity estimates to identify missing change history in change history based recommender (CHB) systems. For this goal, we selected Ying et al.'s approach [41] as a representative of CHB recommender systems. While there are several existing approaches which employ code history to make predictions, we chose Ying et al.'s approach for 2 reasons. Ying et al.'s work was published a few months before the inception of this thesis and used open source projects for evaluation, which enabled us to replicate their experiments. Ying's approach was originally part of her Master's thesis [42] and thus the scope and depth of her work matched that of this thesis.

In retrospect, Zimmermann et al.'s CHB approach [44], implemented through their tool ROSE, is also a relevant baseline approach for our work. However, their work was much more detailed (part of Zimmermann's PhD work) and a re-evaluation of their work would have been out of scope for this thesis.

In the end, we realize that our choice of baseline approach guides and dictates much of the work of this thesis. Still, the question raised by this thesis, namely, "Can we substitute missing history in CHB recommender systems using similarity estimates to increase the scope

of CHB recommender systems?" is relevant to all CHB recommender systems with less than 100% recall rate.

7.2 Experimental Setup as an Approximation of Real World Usage of Recommender Systems

For evaluation of our recommender approach,we follow Ying et al.'s technique of checking the recommendations provided by our approach on some finished modification tasks. We discuss some of the unique challenges presented by our evaluation setup and their effects on the results.

7.2.1 Fixed versus sliding training data

To assess the usefulness of a recommender approach, the availability of *correct* recommendations to compare against those supplied by the recommender approach is essential.

One way of evaluation is asking experienced developers working on real modification tasks to use a recommender approach and compare the results against those intended. While this is a good setup for evaluation, it is difficult to execute if the requirement is to compare several recommender approaches for a large number of test cases (modification tasks in our case). Another difficulty is that often developers might themselves not know all the changes required to fix a reported bug. When identifying valid solution sets for modification tasks during our evaluation, we observed that more than 50% of the modification tasks were fixed over 2 or more commits. Some of these multiple commits were separated by more than 12 hours. This implies that for at least some modification tasks developers might not know all the involved entities (files and methods) at the beginning of the task. Lastly, this type of testing is very subjective and depends heavily on the developers' inherent knowledge and experience in the software system. Executing our experiments after-the-fact ensures, to some extent, that we do not encounter the oracle problem and have a valid solution set to compare our results against.

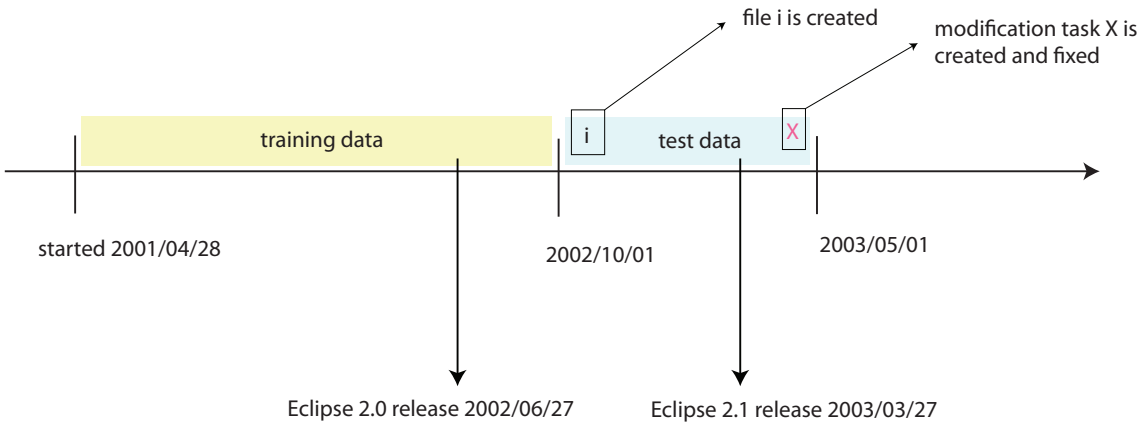

Figure 7.1: Eclipse timeline depicting resolution time of task m and creation time of entity i.

However, engineering an experimental setup to use modification tasks already fixed also mean that we approximate the real world scenario. For example, in the evaluation of their original approach Ying et al. fixed a training data and a test data period. Frequent patterns were extracted through the training data period and evaluated against modification tasks selected in test data period.

Now imagine a modification task m with a file i as part of its solution set. Figure 7.1 shows the test and training data timeline in addition to time of resolution of m and time of creation of i. The modification task is created and resolved towards the end of the test data timeline whereas file i is created at the beginning of the test data timeline. File i may have significant history from its creation till its modification as part of task m. However, due to fixed end points for training data timeline, file i's history will not be considered.

A substitute to the fixed training data period is sliding training data where, for a given modification task, entire history of the system *till* the modification task creation can be used to make recommendations. However, this also implies that frequent patterns for the history under consideration would have to be generated on the fly. A fixed training data period allows us to extract frequent patterns in advance making our recommendation process faster.

7.2.2 Selection of Eclipse versions for ECHB

The selection of the two versions is a tricky question. In real time, the older version should be immediately after the end of training data and the newer version should be the version in which the modification task is being attempted. But, to do such a testing for multiple tasks spread over many versions is beyond the scope of the thesis. In addition, it was important to select two major releases so as to have adequate number of transformations between the two versions.

Consider the ideal case of the older version being immediately at the end of training data and the newer version being the current version. Let x be a file which has accumulated history in the training data period and let it be transformed to x' after the training data. If this file x' is part of a solution set, then conventional CHB recommender systems will provide no recommendations when x' is given as input. But, because of the placement of our two versions, one before and one after the transformation of x to x', our ECHB approach will be able to relate x' to x and obtain the history of x. Similarly, consider another file a that got transformed to a' after the end of training data and let a be returned as a recommendation by conventional CHB recommender systems. But, since a has been transformed, this recommendation is of no use to a developer. But, our ECHB approach will be able to track the transformation and will return a' as the correct recommendation. Thus, the ideal time of release of the two versions between which we will detect transformations has a lot of advantages. But, since we want to test our approach against multiple modification tasks which were fixed during different intermediate versions of Eclipse, it is imperative we choose one version of Eclipse that was released after all (or most modification tasks) were completed. Another constraint in the selection of the two versions is that both of them need to be major releases. This ensures that there will be sufficient number of transformations between the two versions as code transforming activities are part of major releases.

So, for our approach, we selected two releases of Eclipse, R2_1 and R2_0. Release R2_1 is

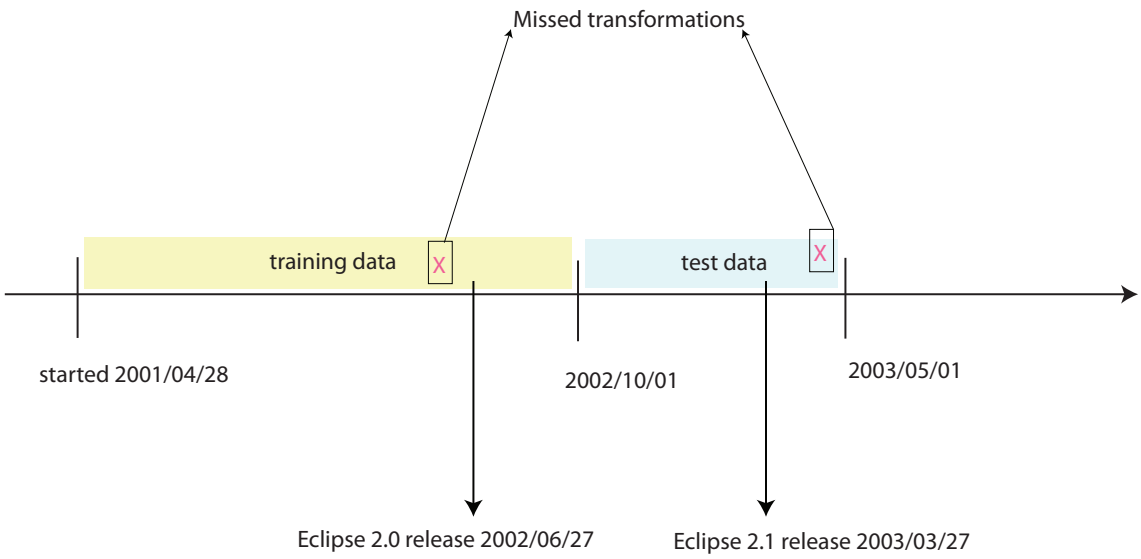

Figure 7.2: Eclipse timeline shows the time period during which if transformations occur our approach would miss it.

towards the end of training data and release R2_0 is towards the end of test data. Eclipse team also publishes releases notes [1, 2] for each release giving a high-level overview of the magnitude and amount of changes for a given release. In addition Xing and Stroulia [40] studied the structural evolution of Eclipse and their findings for various versions of Eclipse, including releases R2_1 and releases R2_0. Thus, various sources indicate that R2_1 and R2_0 were important releases for the Eclipse IDE that involved significant number of changes. Hence, these two releases were best suited to study the efficacy of our transformation detection extension.

Though our choice of the two releases to detect transformations have many advantages and in our opinion are best suited for our testing, they are not without their share of disadvantages. There are locations in the Eclipse timeline where if transformations had occurred, we would have missed detecting them. Figure 7.2 shows the points in time, when, if a transformation occurs our experimental setup would miss it. These points are represented through symbol "X".

In the end, our evaluation set up is an approximation and just like Ying et al.'s original approach, in a real world scenario a developer can easily select the appropriate older release

that would act as the initial release against which entities in the current version would be compared. Selecting fixed releases allowed us the flexibility to extract various data a priori making it easier to carry out detailed experiments quickly. In addition, in Sections 6.3 and 6.4 we discuss the exact conditions under which our extension would provide an improvement over the original CHB approach. We also carry out experiments evaluating the inputs that satisfy these conditions so as to inform the readers how our approach would perform in real world scenario despite missed transformations due to selected evaluation timeline.

7.3 Usage Scenarios for the Six Approaches Compared

Based on various experimental results for the 6 approaches under consideration we make some suggestions for the usage of individual approaches.

- **CHB approaches**: Our re-evaluation of CHB approaches for two granularities showed that this approach is indeed useful in recommending relevant entities to developers working on modification tasks. Some of these entities cannot be identified as pertinent by the current static and dynamic approaches. Even for cases where help exists in the form of various tools for software structure evaluation and human knowledge in the team, recommendations from CHB approaches can act as important supplementary information. However, to get the best recommendations, selection of an appropriate frequency threshold (ω_{method}, ω_{file}) is crucial. We found that for both file-level and method-level implementations of CHB, even the lowest threshold of 2 resulted in significant number of useful recommendations. Perhaps, in future, enhancements of CHB can use a technique similar to Zimmermann et al. [44] and use a low frequency threshold in conjugation with the support threshold (number of time any two entities changed together) and suggest top 10 recommendations with highest support threshold after filtering out recommendations below the lowest frequency threshold of 2. Thus CHB approach can be used for systems

with moderate to significant history.

- **SB approaches**: Through our evaluation, we found that entity similarity alone do not result in useful recommendations. Perhaps, a variant of our SB approaches can be used as a tool for monitoring and studying software evolution. This can be achieved by extending our CREATE-SIMILARITY-GRAPH for multiple versions allowing developers to see the effects of transformation on a single entity (or multiple entities) over various versions. Low precision and recall values for our SB approach was a further testament to the usefulness of CHB approaches.

- **ECHB approaches**: The results of our ECHB approach show that for method-level our approach provides valid results in cases where CHB does not. We found that in as many as 50% of the cases when a transformation occurred our extended ECHB approach provided valid recommendation. While the precision and recall of results in these cases is low we think that our approach can be useful in its current form for systems which undergo transformations constantly. We carried out two variants of our ECHB approach and found one (BM) to be more effective than the other (TH). Further research is needed in identifying ways in which we can modify our similarity algorithm and our technique for combining CHB and SB to improve the results from the ECHB approach.

7.4 Limitations of Godfrey and Zou's Origin Analysis Technique

Our transformation detection approach is based on Godfrey and Zou's work on origin analysis to detect merging and splitting [45]. Their work is implemented through the tool Beagle. Beagle uses "matchers" or matching modules implementing different matching techniques. The four matchers used in their work are *name_matcher* (same as our method name fact comparison), *declaration_matcher* (similar to our return type and parameter comparison), *relation_matcher* (similar to our caller and callee comparison) and *metrics_matcher* using the Kon-

togiannis metrics [29] (we have not implemented this).

Thus, our approach is largely based on Godfrey's origin analysis technique. However, unlike our approach their approach is not completely automatic. Beagle tool performs origin analysis through the following steps:

1. User selects a set of entities from the previous version to be compared, known as "candidate entities" against the "target" entity from the later version.

2. The user applies one or more matchers to compare the "target" and "candidate" entities.

3. Beagle uses the results from various matchers and ranks the "candidate" entities according to the results from the matchers.

It is evident that Godfrey and Zou's origin analysis technique requires user input in the form of selection of "target" and "candidate" entities and later for selection of one or matchers. Thus, their approach relies on the fact that the user already has knowledge of potential "candidates" for the "target" entities under comparison. However, their assumption may not apply for systems as large as Eclipse or for developers working on existing code base.

To automate similarity detection, we have applied a fixed combination of various facts in addition to treating the set of all entities as a "candidate" set. It is difficult to give an absolute measure of correctness of the origin analysis technique or our similarity detection algorithm as there is no well accepted *best* way to detect similarities and the usefulness of the similarity measure depends on the use case. Our technique has its strengths and weaknesses.

7.5 Threats to Validity

We examine the soundness of our approach by discussing threats to its validity in this section. In this section, we discuss threats to *internal validity* and *external validity* of our approach.

Internal Validity Internal validity examines if the results of our experiments are affected by any spurious relationships in addition to the relationships that we have studied.

We hypothesize that using knowledge of transformations, we can identify missing history and hence improve CHB recommender approaches. The primary threat to the construct validity of our approach is that it is difficult to distinguish transformation from similarity. For example, consider entities A and B that are highly similar in version $n-1$ and without undergoing any transformation retain their similarity in version n. Now, our approach would detect similarity between entity B in version n with entity A in version $n-1$ and vice versa, and report it as a transformation even though a transformation did not occur. To counter this, we have performed a specific experiment of our ECHB and CHB approach (see Section 6.3) where we use only those entities as input which exist in version n but not in $n-1$. This partially eliminates the confounding factor of similarity without transformation.

External Validity External validity examines if the results of our experiments are generalizable. We have used only one system, Eclipse, as our benchmark system and, without further research, it is difficult to predict the usefulness of our results for other systems. Eclipse is a complex real world system with multiple developers and we used all 419 projects in the code base for the evaluation experiments. The nature of our work allowed detailed experiments for only one complex system. Nevertheless, we believe that software evolution is a common denominator for all non-defunct projects and transformations are a natural result of many evolutionary activities like refactoring. Thus, our approach should work for systems beyond Eclipse. The results might differ for different systems.

In addition, we have only extended Ying et al.'s CHB approach and hence we cannot argue that our work would perform similarly for all CHB approaches. However, as discussed, transformations are a common property of evolving systems and hence any CHB approach that does not take them into account would perform poorly when an entity of interest has been

transformed. In these cases our extended approach can provide better results.

7.5.1 Summary

In this chapter, we summarize some issues about our approach and its evaluation. In Section 7.1, we explain the reasons for selecting Ying et al.'s CHB approach as a baseline approach for our work. Section 7.2 describes some of the challenges arising due to after-the-fact evaluation of the modification tasks already fixed. In Section 7.3, we describe some of the usage scenarios where individual approaches evaluated in this thesis may perform better compared to others. We have used Godfrey and Zou's origin analysis technique as a basis of our technique for detecting method-pair similarity. In Section 7.4, we discuss some of the limitations of their technique which can affect the quality of our results. Finally, Section 7.5 discusses the threats to the validity of our work.

Chapter 8

RELATED WORK

In our best knowledge, no change history mining based approaches include transformation information when making predictions. So, in this chapter, we compare and contrast our work with different classes of change history based recommender approaches. The three basic approaches are: (a) techniques that rely on development history (See Section 8.1) (b) techniques that use code similarity (See Section 8.2) (c) techniques that help in understanding the evolutionary nature of software systems (See Section 8.3).

8.1 Using Change History to Detect Relevant Code

Gall and colleagues were some of the earliest researchers to explore the use of change history for detecting logical coupling among program entities [18]. To this end, they developed a technique called CAESAR using *Change Sequence Analysis* to identify change patterns in a system. The detected logical couplings are then verified using *Change Report Analysis*. Change reports consist of a detailed description of a given change in system, along with the participating entities for the change. If Change Reports consist of the Change Patterns as suggested by CAESAR, then the technique is deemed useful for detecting hidden logical coupling. While our approach builds on the idea of change patterns in the form of frequent patterns, our work differs from the work of Gall et al. in several respects. We have extended the basic change history mining approach to include transformation information. Our work is predictive and is used for predicting entities of interest involved in a modification task; while CAESAR is used to detect logical couplings with the intent of understanding the logical couplings.

Our work extends the approach proposed by Ying et al. and, expectedly, is most similar to

their approach. Ying et al. use frequent pattern mining on change history to make predictions of potential interest. However unlike our approach their approach does not track similarity of code over refactored and transformed versions. In addition our extended approach works at both method-level and file-level granularity as opposed to only file-level granularity of Ying et al.'s approach. Results of our quantitative experiments show that our extended approach is able to provide recommendations in as many at 50% of the cases where the input is a potentially transformed entity.

Zimmermann et al. developed a tool named ROSE that used association rule mining to recommend related changes given a fragment of code. The primary difference between ROSE and our approach again is that our approach identifies and includes transformation information when making predictions while ROSE only relies on change history when making predictions. In addition, Zimmermann's approach can provide recommendations on the fly, whereas our approach requires extraction of frequent patterns and transformational information beforehand.

Canfora and Cerulo [13, 14] mine the textual description of a bug fix to guide developers working on a new bug fix. The rationale behind their approach is that developers working on new bug fixes might benefit from older bug fixes having similar description. The disadvantage of this approach is the reliance on textual description of a bug fix, which may or may not be accurate in many cases and our approach is not restrained by this.

Similarly other approaches leveraging change history [43, 34] do not account for name and/or location modifying transformations; while relying on name and location for establishing the identity of source code entities. While this issue has been recognized in Ying et al. and Zimmermann et al.'s work [41, 44] it has not been incorporated in any of the newer works. Our approach is the first approach to incorporate the knowledge of transformations with a change history based approach to make transformation aware recommendations.

8.2 Using Code Similarity to Detect Relevant Code

In addition to change history mining approaches various similarity based approaches have been used by developers to guide them in modification tasks. These approaches are based on the hypothesis that looking at similar code can help developers in a particular change task. The developer provides a fragment of code as input and other portions of code similar to the input are returned as result.

As our ECHB approach relies on code similarity to detect transformations, it is useful to compare and contrast some of the existing similarity based recommender approaches with our approach for detecting transformations. Hipikat [15] and Strathcona [23] are two such tools. While Hipikat uses artifacts in addition to code to recommend relevant information, Strathcona makes use of existing similar code fragments to inform developers. Both these approaches make use of heuristics to determine code similarity as a basis of recommendations.

Hipikat makes suggestions based on the textual similarity between the description of modification task at hand and various other artifacts in a system; like bug reports in Bugzilla, source code files, file revisions stored in Version Control Systems, email messages and newsgroup messages. In this respect, Hipikat uses a wider variety of information compared to our approach. However, Hipikat does not take transformation information into account. It also requires a similar modification task to have occurred in the past and thus may not be useful for completely new modification tasks.

Strathcona helps users of an Application Programming Interface (API) by suggesting example code fragments from repository of source code using the API. Strathcona uses various heuristics (Inherits, Calls, Uses and References) to match a given code fragment with relevant examples. Our approach is similar to Strathcona as our ECHB approach also uses similar facts as those identified by the heuristics (Callers, Callees, Parameters, Return Type and Name) to identify similarity. However our goal is to identify transformations between versions as

opposed to similar code fragments in current version of systems using a particular API. Our approach also builds on top of Ying et al.'s approach thus effectively using both change history and transformation information. Ultimately Strathcona and our ECHB approach are addressing different problems.

8.3 Other Approaches in Understanding Software Evolution

Our ECHB approach detects transformations over versions of a software system and in this respect it is similar to some of the approaches which are aimed at understanding the evolutionary nature of a system over various versions. In this section we focus on these approaches.

Godfrey and Zou [45] use origin analysis - the technique of identifying origins of various source code entities to better understand the original intent of some of the design changes. Our ECHB approach's five facts used in detection of transformations (callers, callees, parameters, return type and name) is based on the facts used in origin analysis. Origin analysis however uses a more rigourous measure for establishing origin of entities by including Kontogiannis [29] metrics in addition to the above mentioned facts. For our purposes we need to keep the detection lightweight and hence we have not used Kontogiannis metric. Godfrey and Zou's technique requires the developer to provide a set of candidate and target entities which are then compared for transformational links. Our ECHB approach on the other hand does not require any developer input to detect transformations.

S. Kim et al. [28] used the technique of origin analysis as suggested by Godfrey and Zou; but automated the detection of original entities by combining various similarity measures in a weighted manner. In this respect our ECHB approach's detection of similarity is similar to their approach. However, unlike our approach they do not use the detected origin relationships for any predictive work.

Kim and colleagues investigated clone life cycles using their clone genealogy tool which

automatically extracts clone information from Version Control Systems [27, 25]. They discovered that it is not always easy to refactor clones in a system and a significant number of clones resolve themselves with time. While their work focuses solely on how clones evolve over versions we are interested in software entities that have undergone transformation loosing past history. We are also more interested in using the identified transformation in making predictions for developers working on modification tasks.

Kim et al. [26] have developed a tool to determine and represent structural changes in a given system over various versions in the form of concise rules. Their approach is similar to our ECHB approach in the way of detection of transformations over any two versions. However, our approach is different from their tool in two ways. While their tool supports a pre-specified set of rules and determines a subset of these rules, our tool detects transformations on the basis of similarity. Thus while their tool might report more accurate results with respect to the said transformations; our tool would identify a larger section of transformation. Further more while the goal of their work is to determine and concisely represent transformations we use the detected transformational information to buttress change history based recommender systems.

JDiff by Apiwattanapong et al. [6, 8] compares two versions of a Java program based on an enhanced control flow graph (ECFG). JDiff is an extension of Laski and Szermer's algorithm [30] for comparing control flow graph (CFG) nodes in two programs. The algorithm reduces a CFG into a *hammock* or a single entry, single exist subgraph. Next the algorithm (and JDiff) compares the hammocks in a depth first manner. JDiff is an improvement on Laski and Szermer's original work in terms of performance and robustness. JDiff uses the top-down approach to find class-level, method-level and node-level correspondence through the same algorithm and in this respect it is more effective than our algorithm which find only method-similarity and than builds on it to find class-level similarity. However, while JDiff has been used in regression test selection [36] and dynamic impact analysis [7] our approach is the only one which uses the detected method and class level correspondence to fill in historical gaps in

a CHB approach.

8.4 Summary

In this chapter we have discussed some of the research pertinent to this thesis. All the examined related work can be classified under three categories: research using change history to detect relevant code, research using code similarity to detect relevant code and other approaches in understanding software evaluation. While all three classes of work bear resemblance to our work, none of the existing approaches to our knowledge use transformation information in addition to change history of a system to provide recommendations to developers working on modification tasks.

Chapter 9

CONCLUSIONS AND FUTURE WORK

In this thesis, we have described our similarity based transformation detection approach as an extension to Change History Based (CHB) recommender approaches. Current CHB approaches cannot provide good recommendations in case of modification tasks that involve source code entities that have little or no history. We hypothesized that by filling in this gap we can improve the scope of CHB approaches. To test our hypothesis, we selected Ying et al.'s CHB approach as a representative of existing CHBs and extended it. We conjectured that our approach could provide useful recommendations where current CHB approaches fail due to lack of history for a newly created source code entity or missing history for transformed source code entities.

Since Ying et al.'s CHB approach is not available as a tool, to extend it, we first re-implemented their approach from scratch. Re-implementation of their approach proved to be complex due to missing/ambiguous details about their work. We also repeated their evaluation experiments to attest the correctness of our implementation. Of the two benchmark systems, Eclipse and Mozilla, we repeated their experiment for Eclipse as our intended extension to their work can work only for Java programming language currently and Mozilla is implemented in C/C++.

We used 20 modification tasks from Eclipse to compare the quality of recommendations provided by various recommender systems. Each file in the solution set of the modification task was provided as input to the CHB recommender system. The obtained recommendations were compared against the remaining files in the solution set. We assess the efficacy of the recommender system using three measures: $\mathrm{pr}'_{\mathrm{avg}}$ for accuracy, $\mathrm{rc}'_{\mathrm{avg}}$ for completeness and $\mathrm{tp}'_{\mathrm{avg}}$ for depicting the fraction of inputs for which the recommender system provides results.

Neither could we obtain all the details necessary to accurately reproduce Yings experiment accurately from their journal publication [41] nor from Ying's Master's thesis [42]. So, we had to perform several variations of their evaluation experiments to arrive at the same results as Ying et al. Each variation was a hypothesis of what Ying had actually performed. Details of their work was compiled from their journal publication [41], Ying's Master's thesis [42], our email correspondence, and a teleconference with Ying. In addition, to test their approach for a range of frequent thresholds, we repeated their experiments for frequent thresholds 2–30. Eventually, we obtained results which were close to Ying et al.'s results for the 9 modification tasks for which they report valid results. However, we also obtained results for some of the other 11 modification tasks for which Ying et al. did not obtain any results. Nevertheless, we built our extended approach (ECHB) using our implementation of Ying et al.'s original approach as baseline.

In addition to re-implementing Ying et al.'s original approach for class-level granularity, we also implemented their approach for method-level granularity. Our goal was to examine the quality of recommendations for both CHB and ECHB approaches for two different granularities. After porting their approach to method-level granularity, we repeated the evaluation experiment for the method-level granularity. We found that for thresholds 2–5 (these thresholds give the best values for the three measures of $\mathrm{pr}'_{\mathrm{avg}}$, $\mathrm{rc}'_{\mathrm{avg}}$ and $\mathrm{tp}'_{\mathrm{avg}}$[1] overall), method-level CHB approach provided more accurate results while the file-level CHB provided more complete results. We believe that depending on the task at hand, both method-level and file-level CHB approaches can be useful to developers.

Next, to test our hypothesis, we implemented ECHB, an extended CHB approach leveraging entity similarity to provide recommendations for transformed entities. We developed a lightweight algorithm (CREATE-SIMILARITY-GRAPH) to detect method-level transformations

[1] At higher frequent thresholds, precision for file-level CHB results is higher than for method-level results, but $\mathrm{tp}'_{\mathrm{avg}}$ is very low and hence we consider only the results up to threshold 5 for both granularities.

amongst two versions of a system. Due to the involvement of a large number of entities, the algorithm needed optimization. So, we developed a lightweight solution by ignoring entities that had retained their name and location. Those entities that were ignored represented untransformed entities. Thus, the set of candidate entities that were compared was reduced by 45%. Next, we leveraged the algorithm CREATE-SIMILARITY-GRAPH to find class-level similarity by combining the method-pair similarities.

To evaluate our ECHB approach we chose two major versions of Eclipse, R2_1 and R2_0 as input for CREATE-SIMILARITY-GRAPH. For each input that did not result in any recommendation through the CHB approach, we found its similar counterpart in version R2_0 of Eclipse using the similarity algorithm. The similar entities are in turn provided as inputs to CHB to find recommendations. In this way, we substitute the missing recommendations of a potentially transformed entity using the recommendations obtained using the original entity. We implement the ECHB approach for two variants, the best-match (BM) and the threshold (TH) variants and repeat the evaluation experiments for both the granularities using the 2 variants. The results from file-level experiments are not very strong. However, the results from the method-level variant showed an improvement in $\mathrm{tp}'_{\mathrm{avg}}$ (with a drop in $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$) for the BM variant. To investigate further, we perform an additional experiment for the BM variant for method-level granularity. We selected only the inputs which satisfy the condition of a transformation and use them for the experiment. This helps in removing some of the confounding factors and shows how our approach fares with respect to CHB for the intended scenario of transformed entity, when used as an input. We find that in about 50% of the cases our ECHB approach was able to provide recommendations when the input entity was potentially transformed, the CHB approach on the other hand provided results only for 14% cases. However the $\mathrm{pr}'_{\mathrm{avg}}$ values for CHB approach were significantly higher than that of ECHB in this experiment (0.47 and 0.09 respectively). The $\mathrm{rc}'_{\mathrm{avg}}$ values were also slightly higher for CHB approach (0.18 and 0.13 respectively).

Finally, we applied our ECHB approach for method-level granularity for the BM variant to substitute missing recommendations, in addition to missing history. The results remained the same as that of our initial experiments.

In addition, to ECHB approaches, we implemented CREATE-SIMILARITY-GRAPH as an independent similarity based (SB) recommender approach to provide recommendations based solely on similarity. The results for both granularities for this standalone SB approach are very weak reinforcing the knowledge that structural similarity alone cannot always constitute useful recommendations.

To conclude, while current CHB approaches leverage change history to provide meaningful recommendations, many of them rely on fixed identifiers for source code entities, ignoring the fact that the identifiers themselves are subject to change. In this thesis, we remedy this assumption of fixed identifiers by introducing our transformation detection approach as an extension to CHB approach to fill missing historical gaps. The results from our ECHB approach are not very strong. However, in specific cases ECHB for method-level provides results when CHB does not. Thus, increasing the scope of inputs where CHB can provide pertinent recommendations. We believe that further research in the area would reveal ways in which ECHB can provide more accurate recommendations when CHB approaches fail. Our approach can be particularly useful for frequently modified systems where source code entities do not have sufficient history to be leveraged by CHB approaches.

9.1 Future Work

We have shown through this thesis that ECHB approach can provide recommendations in scenarios where CHB does not. However, we found that the $\mathrm{pr}'_{\mathrm{avg}}$ and $\mathrm{rc}'_{\mathrm{avg}}$ of values of our ECHB approach were low for these cases. Thus, additional research is required both on the approach and evaluations before our ECHB approach can result in significant gains in recommendation

quality over CHB.

Currently, our system has a single approach for all types of modification tasks. It would be interesting to analyze and classify modification tasks into different categories such that each category will be approached differently by the recommender system. This approach can exploit the nature of the modification task to use the recommender approach best suited for it.

In addition, further research is needed to determine a more accurate similarity measure for both method-level and class-level transformation detection. A deeper understanding into the interplay of the role of various method facts in the computation of the similarity measure could help in devising the best similarity measure. A more sophisticated similarity measure, based on this understanding, will improve the quality of recommendations.

One research direction advancing our work could be to test how our approach fares for transformations over several versions. Technically, our approach can detect similarities over multiple version. But, in our evaluation, we detected similarities only over two consecutive versions.

Finally, we think case studies with developers working on modification tasks in real scenarios can help us refine our ECHB approach and assess the strength of our work more realistically.

9.2 Contributions

This thesis makes three significant contributions in the fields of software evolution.

1. We partially replicate Ying et al.'s FP-tree mining recommender approach to use it as baseline. In the process we assess the strengths and weaknesses of their current approach. In addition, our work shows that recreating an implemented approach an its evaluation is hard. Seldom in the field of Software Engineering evaluation experiments are recreated and this has been a cause of criticism towards Software Engineering. Our work is an example of systematic replication of an existing work and the difficulties encountered in

the process.

2. We also implement Ying et al.'s original approach at method-level granularity and compare and contrast the effects of granularity on efficacy of their approach

3. Last we introduce a lightweight algorithm to detect transformations over 2 versions of a system and use the resulting information as a means to enhance Ying et al.'s CHB approach. We evaluate our ECHB approach and compare it against Ying et al.'s original approach.

Appendix A

DETAILED DATA

A.1 File-Level Solution Sets for Modification Tasks

#24635
org.eclipse.platform.source-feature/feature.xml
org.eclipse.platform.solaris.motif-feature/feature.xml
org.eclipse.platform.linux.motif.source-feature/feature.xml
org.eclipse.platform.qnx.photon.source-feature/feature.xml
org.eclipse.platform.aix.motif-feature/feature.xml
org.eclipse.platform.solaris.motif.source-feature/feature.xml
org.eclipse.platform-feature/feature.xml
org.eclipse.platform.hpux.motif-feature/feature.xml
org.eclipse.platform.linux.gtk-feature/feature.xml
org.eclipse.platform.linux.gtk.source-feature/feature.xml
org.eclipse.platform.qnx.photon-feature/feature.xml
org.eclipse.platform.win32.source-feature/feature.xml
org.eclipse.platform.linux.motif-feature/feature.xml
org.eclipse.platform.win32-feature/feature.xml
org.eclipse.platform.hpux.motif.source-feature/feature.xml
org.eclipse.platform.aix.motif.source-feature/feature.xml
org.eclipse.releng/maps/feature.map
org.eclipse.sdk.tests-feature/feature.xml
org.eclipse.sdk.examples-feature/feature.xml
org.eclipse.sdk.linux.gtk-feature/feature.xml
org.eclipse.sdk.linux.motif-feature/feature.xml
org.eclipse.sdk.qnx.photon-feature/feature.xml
org.eclipse.sdk.aix.motif-feature/feature.xml
org.eclipse.sdk.examples.win32-feature/feature.xml
org.eclipse.sdk.hpux.motif-feature/feature.xml
org.eclipse.sdk.solaris.motif-feature/feature.xml
org.eclipse.sdk.win32-feature/feature.xml
org.eclipse.releng/maps/releng.map
org.eclipse.team.extras-feature/feature.xml

#21330

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/forms/DetailsForm.java

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/model/ConfiguredFeatureAdapter.java

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/model/MissingFeature.java

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/model/PendingChange.java

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/views/ConfigurationView.java

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/wizards/InstallWizard.java

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/wizards/TargetPage.java

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/UpdateUIPlugin.java

#24657

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider.java

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference.java

org.eclipse.update.core/src/org/eclipse/update/core/JarContentReference.java

org.eclipse.update.core/src/org/eclipse/update/core/Utilities.java

org.eclipse.update.core/src/org/eclipse/update/internal/core/InstallConfiguration.java

org.eclipse.update.core/src/org/eclipse/update/internal/core/UpdateManagerLogWriter.java

#25041

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/ClassFile.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CommitWorkingCopyOperation.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CompilationUnit.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CopyResourceElementsOperation.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CreateCompilationUnitOperation.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CreatePackageFragmentOperation.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/DeleteResourceElementsOperation.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/DeltaProcessor.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/JarPackageFragmentRoot.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/JavaProject.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/JavaProjectElementInfo.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/NameLookup.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/Openable.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/PackageFragment.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/PackageFragmentRoot.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/SearchableEnvironment.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/SourceRefElement.java

#25041 (cont'd)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/Util.java

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/WorkingCopy.java

org.eclipse.jdt.core/search/org/eclipse/jdt/core/search/SearchEngine.java

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/DeclarationOfAccessedFieldsPattern.java

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/DeclarationOfReferencedMethodsPattern.java

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/DeclarationOfReferencedTypesPattern.java

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/MatchLocator.java

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/SuperTypeNamesCollector.java

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/HierarchyScope.java

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/JavaSearchScope.java

build-notes file

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/DeleteElementsOperation.java

#13907

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/compare/JavaTokenComparator.java

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Parser.java

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Scanner.java

org.eclipse.jdt.core/formatter/org/eclipse/jdt/internal/formatter/CodeFormatter.java

build-notes file

#23096

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Parser.java

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Scanner.java

org.eclipse.jdt.core/formatter/org/eclipse/jdt/internal/formatter/CodeFormatter.java

build-notes file

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTTest.java

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/AST.java

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/ASTConverter.java

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/Statement.java

#24668

org.eclipse.core.runtime/src/org/eclipse/core/internal/plugins/PluginDescriptor.java

#23587

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/ASTResolving.java

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/CorrectionMessages.properties

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewCUCompletionUsingWizardProposal.java

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewMethodCompletionProposal.java

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal.java

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/SimilarElementsRequestor.java

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor.java

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/JavaPluginImages.java

#24730

org.eclipse.team.cvs.core/src/org/eclipse/team/internal/ccvs/core/connection/CVSRepositoryLocation.java

org.eclipse.team.cvs.core/src/org/eclipse/team/internal/ccvs/core/ICVSRepositoryLocation.java

org.eclipse.team.cvs.ui/src/org/eclipse/team/internal/ccvs/ui/wizards/RestoreFromRepositoryFileSelectionPage.java

#16817

org.eclipse.jdt.debug/model/org/eclipse/jdt/internal/debug/core/breakpoints/JavaBreakpoint.java

org.eclipse.debug.core/core/org/eclipse/debug/core/model/Breakpoint.java

#24165

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/sync/CatchupReleaseViewer.java

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/IHelpContextIds.java

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/messages.properties

#24406

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/DefaultBindingResolver.java

build-notes file

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest2.java

org.eclipse.jdt.core.tests.model/workspace/Converter/test0418/A.java

#24424

org.eclipse.debug.core/core/org/eclipse/debug/core/model/Breakpoint.java

#24424 (cont'd)

org.eclipse.jdt.debug.tests/tests/org/eclipse/jdt/debug/tests/AbstractDebugTest.java

org.eclipse.jdt.debug.tests/tests/org/eclipse/jdt/debug/tests/ProjectCreationDecorator.java

build-notes-jdt-debug file

#24449

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/DefaultBindingResolver.java

build-notes file

#24594

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner.java

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests.java

#24622

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/ASTConverter.java

build-notes file

#24756

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner.java

#24828

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletArgumentsTab.java

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab.java

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletParametersTab.java

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/internal/debug/ui/launcher/JavaAppletLaunchShortcut.java

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/internal/debug/ui/launcher/JavaAppletTabGroup.java

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/internal/debug/ui/launcher/LauncherMessages.properties

org.eclipse.jdt.debug.ui/plugin.properties

build-notes-jdt-debug file

#25124
org.eclipse.jdt.core.tests.model/workspace/Converter/test0447/A.java
org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Parser.java
org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest.java
org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest2.java
org.eclipse.jdt.core.tests.compiler/src/org/eclipse/jdt/core/tests/compiler/parser/DietRecoveryTest.java
build-notes file

#25133
org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner.java
org.eclipse.ant.tests.core/test plugin/org/eclipse/ant/tests/core/testplugin/ProjectHelper.java
org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests.java
org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/AbstractAntTest.java
org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/ProjectCreationDecorator.java
org.eclipse.ant.tests.core/.classpath

A.2 Method-Level Solution Sets for Modification Tasks

#24635

Same as solution set for file-level modification task.

#21330

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/forms/DetailsForm/getDoButtonVisibility()

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/forms/DetailsForm/findFeature(VersionedIdentifier, IFeatureReference[])

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/forms/DetailsForm/doButtonSelected()

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/forms/DetailsForm/updateButtonText(boolean)

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/forms/DetailsForm/executeOptionalInstall(MissingFeature)

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/model/ConfiguredFeatureAdapter/getIncludedFeatures()

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/model/MissingFeature/getParentOriginatingSiteURL()

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/model/MissingFeature/getOriginatingSiteURL()

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/model/PendingChange/getTargetSite()

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/model/PendingChange/PendingChange(IFeature, IConfiguredSite)

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/views/ConfigurationView/makeActions()

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/wizards/InstallWizard/preserveOriginatingURLs(IFeature, IFeatureReference[])

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/wizards/InstallWizard/performFinish()

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/wizards/InstallWizard/execute(IConfiguredSite, Object[], IFeatureReference[], IProgressMonitor)

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/wizards/TargetPage/TargetPage(PendingChange, IInstallConfiguration)

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/wizards/TargetPage/getSiteVisibility(IConfiguredSite)

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/UpdateUIPlugin/setOriginatingURL(String, URL)

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/UpdateUIPlugin/getOriginatingURL(String)

org.eclipse.update.ui/src/org/eclipse/update/internal/ui/UpdateUIPlugin/getOriginatingURLSection()

#24657

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider/getPathID(INonPluginEntry)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider/FileFilter/accept(String)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider/validatePermissions(ContentReference[])

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider/getDownloadSizeFor(IPluginEntry[], INonPluginEntry[])

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider/matchesOneRule(String, Map)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider/getInstallSizeFor(IPluginEntry[], INonPluginEntry[])

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider/asLocalReference(ContentReference, InstallMonitor)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider/getPermissions(ContentReference[])

#24657 (cont'd)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureContentProvider/asLocalFile(ContentReference, InstallMonitor)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/getCategories()

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/getSearchLocation()

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/getMatch()

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/setURL(URL)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/retrieveEnabledFeatures(ISite)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/getBestMatch()

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/getVersionedIdentifier()

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/getFeature()

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/FeatureReference(IFeatureReference)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/isDisabled()

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/createFeature(String, URL, ISite)

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/getName()

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/isOptional()

org.eclipse.update.core/src/org/eclipse/update/core/FeatureReference/matches(VersionedIdentifier, VersionedIdentifier, IncludedFeatureReference)

org.eclipse.update.core/src/org/eclipse/update/core/JarContentReference/unpack(File, String, ContentSelector, InstallMonitor)

org.eclipse.update.core/src/org/eclipse/update/core/JarContentReference/unpack(File, ContentSelector, InstallMonitor)

org.eclipse.update.core/src/org/eclipse/update/core/Utilities/createLocalFile(File, String)

org.eclipse.update.core/src/org/eclipse/update/core/Utilities/createLocalFile(File, String, String)

org.eclipse.update.core/src/org/eclipse/update/core/Utilities/mapLocalFile(String, File)

org.eclipse.update.core/src/org/eclipse/update/internal/core/InstallConfiguration/save(boolean)

org.eclipse.update.core/src/org/eclipse/update/internal/core/InstallConfiguration/savePluginPath(ConfiguredSite, IPlatformConfiguration, IPlatformConfiguration)

org.eclipse.update.core/src/org/eclipse/update/internal/core/InstallConfiguration/saveConfigurationFile(boolean)

org.eclipse.update.core/src/org/eclipse/update/internal/core/UpdateManagerLogWriter/getAction(int)

#24657

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/assertCorrespondingResourceFails(IJavaElement)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/assertOpenFails(IOpenable)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/assertUnderlyingResourceFails(IJavaElement)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testCorrespondingResourceNonExistingCompilationUnit()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testCorrespondingResourceNonExistingJarPkgFragmentRoot()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testCorrespondingResourceNonExistingPkgFragment()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testCorrespondingResourceNonExistingPkgFragmentRoot()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testCorrespondingResourceNonExistingProject()

#24657 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testCorrespondingResourceNonExistingType()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testNonExistingClassFile()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testNonExistingCompilationUnit()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testNonExistingPackageFragment()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testUnderlyingResourceNonExistingClassFile()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testUnderlyingResourceNonExistingCompilationUnit()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testUnderlyingResourceNonExistingJarPkgFragmentRoot()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testUnderlyingResourceNonExistingPkgFragment()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testUnderlyingResourceNonExistingPkgFragmentRoot()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testUnderlyingResourceNonExistingProject()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testUnderlyingResourceNonExistingType()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/WorkingCopyTests/testNonExistingCU()

build-notes file

org.eclipse.jdt.core/model/org/eclipse/jdt/core/ToolFactory/createDefaultClassFileReader(IClassFile, int)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/ClassFile/openWhenClosed(IProgressMonitor)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CommitWorkingCopyOperation/verify()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CompilationUnit/buildStructure(OpenableElementInfo, IProgressMonitor)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CompilationUnit/generateInfos(OpenableElementInfo, IProgressMonitor, Map, IResource)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CopyResourceElementsOperation/collectResourcesOfInterest(IPackageFragment)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CopyResourceElementsOperation/createNeededPackageFragments(IPackageFragmentRoot, String, boolean)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CopyResourceElementsOperation/processCompilationUnitResource(ICompilationUnit, IPackageFragment)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CopyResourceElementsOperation/processPackageFragmentResource(IPackageFragment, IPackageFragmentRoot, String)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CreateCompilationUnitOperation/executeOperation()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CreatePackageFragmentOperation/executeOperation()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/CreatePackageFragmentOperation/verify()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/DeleteResourceElementsOperation/deletePackageFragment(IPackageFragment)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/DeleteResourceElementsOperation/processElement(IJavaElement)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/DeltaProcessor/popUntilPrefixOf(IPath)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/hierarchy/IndexBasedHierarchyBuilder/buildFromPotentialSubtypes(String[])

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/hierarchy/TypeHierarchy/isAffectedByJavaProject(IJavaElementDelta, IJavaElement)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/JarPackageFragmentRoot/getUnderlyingResource()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/JavaProject/generateInfos(OpenableElementInfo, IProgressMonitor, Map, IResource)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/JavaProject/getUnderlyingResource()

#24657 (cont'd)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/JavaProjectElementInfo/computeNonJavaResources(JavaProject)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/NameLookup/findPackageFragment(IPath)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/Openable/buildStructure(OpenableElementInfo, IProgressMonitor)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/Openable/openWhenClosed(IProgressMonitor)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/PackageFragment/getNonJavaResources()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/PackageFragment/openWhenClosed(IProgressMonitor)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/PackageFragment/refreshChildren()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/PackageFragmentRoot/getNonJavaResources()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/PackageFragmentRoot/getUnderlyingResource()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/SearchableEnvironment/findTypes(char[], ISearchRequestor)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/SourceRefElement/getCorrespondingResource()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/SourceRefElement/getUnderlyingResource()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/Util/getResourceContentsAsCharArray(IFile)

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/WorkingCopy/updateTimeStamp(CompilationUnit)

org.eclipse.jdt.core/search/org/eclipse/jdt/core/search/SearchEngine/getResource(IJavaElement)

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/HierarchyScope/buildResourceVector()

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/JavaSearchScope/add(IJavaElement)

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/DeclarationOfAccessedFieldsPattern/reportDeclaration(FieldBinding, MatchLocator)

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/DeclarationOfReferencedMethodsPattern/reportDeclaration(MethodBinding, MatchLocator)

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/DeclarationOfReferencedTypesPattern/reportDeclaration(TypeBinding, int, MatchLocator)

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/MatchLocator/buildBindings(ICompilationUnit)

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/MatchLocator/classFileReader(IType)

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/MatchLocator/locateMatches(String[], IWorkspace, IWorkingCopy[], IProgressMonitor)

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/MatchLocator/locatePackageDeclarations(SearchPattern, IWorkspace)

org.eclipse.jdt.core/search/org/eclipse/jdt/internal/core/search/matching/SuperTypeNamesCollector/buildBindings(ICompilationUnit)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/model/ExistenceTests/testCorrespondingResourceNonExistingClassFile()

org.eclipse.jdt.core/model/org/eclipse/jdt/internal/core/DeleteElementsOperation/verify(IJavaElement)

#13907

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/compare/JavaTokenComparator/JavaTokenComparator(String, boolean)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/compare/JavaTokenComparator/getTokenStart(int)

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Parser/checkAndReportBracketAnomalies(ProblemReporter)

#13907 (cont'd)

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Parser/flushAnnotationsDefinedPriorTo(int)

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Scanner/getNextToken()

org.eclipse.jdt.core/formatter/org/eclipse/jdt/internal/formatter/CodeFormatter/format()

build-notes file

#23096

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Parser/checkAndReportBracketAnomalies(ProblemReporter)

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Parser/flushAnnotationsDefinedPriorTo(int)

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Scanner/getNextToken()

org.eclipse.jdt.core/formatter/org/eclipse/jdt/internal/formatter/CodeFormatter/format()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTTest/tLeadingComment(Statement)

build-notes file

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/AST/parseCompilationUnit(char[])

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/AST/parseCompilationUnit(char[], String, IJavaProject)

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/AST/parseCompilationUnit(ICompilationUnit, boolean)

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/ASTConverter/ASTConverter(boolean)

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/ASTConverter/ASTConverter(Map, boolean)

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/Statement/setLeadingComment(String)

#24668

org.eclipse.core.runtime/src/org/eclipse/core/internal/plugins/PluginDescriptor/getPluginClassLoaderPath(boolean)

org.eclipse.core.runtime/src/org/eclipse/core/internal/plugins/PluginDescriptor/createExecutableExtension(String, Object, IConfigurationElement, String)

org.eclipse.core.runtime/src/org/eclipse/core/internal/plugins/PluginDescriptor/getResourceBundle(Locale)

#23587

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/ASTResolving/getBindingOfParentType(ASTNode)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/ASTResolving/getPossibleReferenceBinding(ASTNode)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/CorrectionMessages.properties

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewCUCompletionUsingWizardProposal/fillInWizardPageName(NewTypeWizardPage)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewCUCompletionUsingWizardProposal/getAdditionalProposalInfo()

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewMethodCompletionProposal/apply(IDocument)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewMethodCompletionProposal/evaluateModifiers()

#23587 (cont'd)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewMethodCompletionProposal/getRewrite()

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/apply(IDocument)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/doAddField(AST, SimpleName, ASTRewrite)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/doAddField(CompilationUnit)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/doAddLocal(AST, SimpleName, ASTRewrite)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/doAddLocal(CompilationUnit)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/doAddParam(AST, SimpleName, ASTRewrite)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/doAddParam(CompilationUnit)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/evaluateFieldModifiers(CompilationUnit)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/getRewrite()

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/NewVariableCompletionProposal(String, ICompilationUnit, int, SimpleName, int)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/NewVariableCompletionProposal/NewVariableCompletionProposal(String, ICompilationUnit, int, SimpleName, ITypeBinding, int, Image)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/SimilarElementsRequestor/findSimilarElement(ICompilationUnit, int, String, int)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/SimilarElementsRequestor/findSimilarElement(ICompilationUnit, int, String, int, int, String)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/SimilarElementsRequestor/findSimilarElement(ICompilationUnit, Name, int)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/SimilarElementsRequestor/findSimilarElement(ICompilationUnit, SimpleName, int)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/SimilarElementsRequestor/process(ICompilationUnit, int)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/SimilarElementsRequestor/SimilarElementsRequestor(String, int, int, String)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor/addNewTypeProposals(ICompilationUnit, Name, int, ArrayList)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor/addSimilarTypeProposals(SimilarElement[], ICompilationUnit, Name, ArrayList)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor/addSimilarTypeProposals(SimilarElement[], ICompilationUnit, SimpleName, ArrayList)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor/createTypeRefChangeProposal(ICompilationUnit, String, Name, int)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor/createTypeRefChangeProposal(ICompilationUnit, String, SimpleName, int)

#23587 (cont'd)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor/getConstructorProposals(ProblemPosition, ArrayList)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor/getMethodProposals(ProblemPosition, boolean, ArrayList)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor/getTypeProposals(ProblemPosition, int, ArrayList)

org.eclipse.jdt.ui/ui/org/eclipse/jdt/internal/ui/text/correction/UnresolvedElementsSubProcessor/getVariableProposals(ProblemPosition, ArrayList)

#24730

org.eclipse.team.cvs.core/src/org/eclipse/team/internal/ccvs/core/connection/CVSRepositoryLocation/getRemoteFile(String, CVSTag)

org.eclipse.team.cvs.core/src/org/eclipse/team/internal/ccvs/core/ICVSRepositoryLocation/getRemoteFile(String, CVSTag)

org.eclipse.team.cvs.core/src/org/eclipse/team/internal/ccvs/core/ICVSRepositoryLocation/getRemoteFolder(String, CVSTag)

org.eclipse.team.cvs.ui/src/org/eclipse/team/internal/ccvs/ui/wizards/RestoreFromRepositoryFileSelectionPage/handleFileSelection(IFile)

#16817

org.eclipse.jdt.debug/model/org/eclipse/jdt/internal/debug/core/breakpoints/JavaBreakpoint/removeFromTarget(JDIDebugTarget)

org.eclipse.jdt.debug/model/org/eclipse/jdt/internal/debug/core/breakpoints/JavaBreakpoint/fireChanged()

org.eclipse.debug.core/core/org/eclipse/debug/core/model/Breakpoint/markerExists()

#24165

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/sync/CatchupReleaseViewer/initializeActions(SyncCompareInput)

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/sync/CatchupReleaseViewer/openSelection()

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/sync/CatchupReleaseViewer/OpenAction/OpenAction(String, ImageDescriptor)

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/sync/CatchupReleaseViewer/OpenAction/run()

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/sync/CatchupReleaseViewer/OpenAction/update()

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/sync/CatchupReleaseViewer/fillContextMenu(IMenuManager)

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/sync/CatchupReleaseViewer/openSelection(Object)

org.eclipse.team.ui/src/org/eclipse/team/internal/ui/messages.properties

#24406

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/DefaultBindingResolver/getMethodBinding(MethodBinding)

build-notes file

#24406 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest2/test0418()

#24424

org.eclipse.debug.core/core/org/eclipse/debug/core/model/Breakpoint/delete()

org.eclipse.jdt.debug.tests/tests/org/eclipse/jdt/debug/tests/AbstractDebugTest/setSuspendOnUncaughtExceptionsPreference(boolean)

org.eclipse.jdt.debug.tests/tests/org/eclipse/jdt/debug/tests/ProjectCreationDecorator/testProjectCreation()

build-notes-jdt-debug file

#24449

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/DefaultBindingResolver/getMethodBinding(MethodBinding)

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/DefaultBindingResolver/resolveName(Name)

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/DefaultBindingResolver/getVariableBinding(VariableBinding)

build-notes file

#24594

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/processCommandLine(List)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/processTargets(List)

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testSpecifyTargetAsArgWithOther Options()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testSpecifyTargetsAsArgWithOther Options()

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/getArgument(List, String)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/getArguments(List, String)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/processCommandLine(List)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/preprocessCommandLine(List)

#24622

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/ASTConverter/getOperatorFor(int)

org.eclipse.jdt.core/dom/org/eclipse/jdt/core/dom/ASTConverter/convert(BinaryExpression)

build-notes file

#24756

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/createLogger()

#24756 (cont'd)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/run(List)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/fireBuildFinished(Project, Throwable)

#24828

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/createControl(Composite)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/initializeFrom(ILaunchConfiguration)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/updateProjectFromConfig(ILaunchConfiguration)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/updateMainTypeFromConfig(ILaunchConfiguration)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/performApply(ILaunchConfiguration-WorkingCopy)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/dispose()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/createVerticalSpacer(Composite)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/handleSearchButtonSelected()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/handleProjectButtonSelected()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/chooseJavaProject()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/getJavaProject()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/getWorkspaceRoot()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/getJavaModel()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/isValid(ILaunchConfiguration)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/initializeDefaults(IJavaElement, ILaunchConfigurationWorkingCopy)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/setDefaults(ILaunchConfigurationWorkingCopy)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/initializeMainTypeAndName(IJavaElement, ILaunchConfigurationWorkingCopy)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/initializeDefaultVM(ILaunchConfigurationWork-ingCopy)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/initializeHardCodedDefaults(ILaunchConfiguration-WorkingCopy)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/getName()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/getImage()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/debug/ui/launchConfigurations/AppletMainTab/getProjectOutputDirectory()

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/internal/debug/ui/launcher/JavaAppletLaunchShortcut/createConfiguration(IType)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/internal/debug/ui/launcher/JavaAppletTabGroup/createTabs(ILaunchConfigurationDialog, String)

org.eclipse.jdt.debug.ui/ui/org/eclipse/jdt/internal/debug/ui/launcher/LauncherMessages.properties

org.eclipse.jdt.debug.ui/plugin.properties

build-notes-jdt-debug file

#25124

org.eclipse.jdt.core/compiler/org/eclipse/jdt/internal/compiler/parser/Parser/consumeFieldDeclaration()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0362()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0319()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0246()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0022()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0392()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0130()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0030()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0146()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0108()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0016()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0381()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0074()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0205()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0221()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0040()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0397()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0127()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0264()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0279()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0330()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0107()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0080()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0018()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0220()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0129()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0036()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0027()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0035()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0243()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0331()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0126()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0055()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0338()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0172()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0181()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0136()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0072()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0088()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0104()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0021()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0282()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0203()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0109()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0161()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0233()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0374()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0145()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0017()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0033()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0232()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0390()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0063()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0054()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0034()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0253()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0105()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0277()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0269()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0011()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0313()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/getASTNodeToCompare(CompilationUnit)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0076()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0207()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0266()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0060()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0081()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0192()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0276()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0037()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0342()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0283()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0369()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0183()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0173()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0245()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0318()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0135()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0301()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0038()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0291()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0065()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0174()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0217()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0236()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0223()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0187()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0057()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0164()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0106()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0376()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0222()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0290()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0352()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0061()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0326()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0349()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/getASTNode(CompilationUnit, int)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/getASTNode(CompilationUnit, int, int, int)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0361()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0278()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0340()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0097()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0357()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0235()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0204()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0165()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0297()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0013()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0388()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0121()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0260()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0118()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0251()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0400()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0186()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0250()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0156()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0240()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0062()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0231()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0286()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0385()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0103()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0189()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0259()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0216()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0096()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0049()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0399()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0239()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0014()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0196()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0320()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0180()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0303()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0380()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/runConversion(char[], String, IJavaProject)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0193()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0391()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0053()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0293()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0048()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0258()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0359()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0287()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0084()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0334()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0377()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0056()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0128()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0265()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0190()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0254()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0150()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0353()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0356()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0101()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0339()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0365()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0148()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0047()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0230()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0302()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0091()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0345()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0296()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0073()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0234()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0028()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0092()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0248()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0335()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0378()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0051()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0242()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0255()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0166()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0090()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0178()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0010()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0398()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/getASTNode(CompilationUnit, int, int)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0019()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0311()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0200()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0346()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0191()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0167()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0153()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0256()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0393()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0066()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0284()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0023()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0336()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0308()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0151()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/suite()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0157()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/setupConverterJCL()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0102()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0208()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0280()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0347()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0009()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0185()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0122()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0085()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0294()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0344()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0059()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0068()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0364()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0124()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0005()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0046()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0050()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0093()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0002()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0007()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0314()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0202()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/getConverterJCLRootSourcePath()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0324()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0020()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0262()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0316()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/tearDownSuite()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0045()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0042()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0159()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0197()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0143()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0247()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0095()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0373()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0315()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0024()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0295()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0310()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0184()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/checkSourceRange(ASTNode, String, char[])

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0155()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0043()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0261()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0307()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0044()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0177()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0154()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0006()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0363()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0098()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0325()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0228()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0058()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0079()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0152()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0389()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0226()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0306()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0132()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0069()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0252()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0115()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0395()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0321()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0305()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0012()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0333()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0176()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0218()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0371()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0343()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0052()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0168()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0114()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0182()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0366()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0267()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0113()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/getConverterJCLSourcePath()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0244()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0312()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0094()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0268()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0160()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0211()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0015()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0387()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0210()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/runConversion(ICompilationUnit, boolean)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0309()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0144()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0142()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0394()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0337()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/getConverterJCLPath()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0067()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0275()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0285()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0358()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0158()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0089()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0125()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0227()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0304()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0201()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0209()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0111()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0292()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0329()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0219()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0375()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0140()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0078()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0041()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0083()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0175()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0004()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0225()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0273()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0169()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0075()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0116()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0384()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0070()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0270()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0224()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0100()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0351()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0141()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0133()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0087()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0119()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0263()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0077()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0110()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0179()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0379()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0008()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0029()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0367()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0139()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0147()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0032()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0162()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0123()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0138()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0149()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0194()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0317()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0213()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0368()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0370()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0360()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0212()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0170()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0322()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0327()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0271()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0120()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0026()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0003()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0341()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0299()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0289()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0372()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0064()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0298()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0137()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/isMalformed(ASTNode)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0237()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0300()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0188()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0328()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0163()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0323()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0082()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0195()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0396()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0348()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0117()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0214()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0025()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0039()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0382()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0229()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0249()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0134()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0199()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0288()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0071()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0198()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0031()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0086()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0350()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0112()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0257()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0355()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0272()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0171()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0332()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0383()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0001()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0215()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0099()

#25124 (cont'd)

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0131()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0274()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0386()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0354()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0241()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/setUpSuite()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0238()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0206()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest/test0281()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest2/test0447()

org.eclipse.jdt.core.tests.model/src/org/eclipse/jdt/core/tests/dom/ASTConverterTest2/suite()

org.eclipse.jdt.core.tests.compiler/src/org/eclipse/jdt/core/tests/compiler/parser/DietRecoveryTest/test98()

org.eclipse.jdt.core.tests.compiler/src/org/eclipse/jdt/core/tests/compiler/parser/DietRecoveryTest/test74()

org.eclipse.jdt.core.tests.compiler/src/org/eclipse/jdt/core/tests/compiler/parser/DietRecoveryTest/test87()

build-notes file

#25133

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/loadPropertyFiles()

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/addInputHandler(Project)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/processProperties(List)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/run(List)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/processCommandLine(List)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/preprocessCommandLine(List)

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testSpecifyTargetAsArgAndQuiet()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testSpecifyTargetsAsArgWithOther Options()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testPropertyFileWithNoArg()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testPropertyFileWithMinusDTaking Precedence()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testPropertyFileFileNotFound()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testSpecifyTargetAsArgWithOtherOptions()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testPropertyFile()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/tests/OptionTests/testSpecifyBadTargetAsArg()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/AbstractAntTest/getPropertyFileName()

org.eclipse.ant.tests.core/tests/org/eclipse/ant/tests/core/ProjectCreationDecorator/testProjectCreation()

org.eclipse.ant.tests.core/.classpath

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/addInputHandler(Project)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/processProperties(List)

org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/preprocessCommandLine(List)

#25133 (cont'd)
org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/loadPropertyFiles()
org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/getFileRelativeToBaseDir(String)
org.eclipse.ant.core/src_ant/org/eclipse/ant/internal/core/ant/InternalAntRunner/createLogFile(String)

A.3 Task-wise Results for the Method-Level CHB Approach

Bug ID	$\omega_{\text{file}} = 2$			$\omega_{\text{file}} = 3$			$\omega_{\text{file}} = 4$			$\omega_{\text{file}} = 5$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	0.35	0.87	1.00	0.46	0.87	1.00	0.81	0.84	1.00	0.80	0.70	0.97
21330	0.13	0.04	0.42	0.23	0.02	0.26	0.00	0.00	0.21	0.00	0.00	0.11
24657	0.31	0.05	0.59	0.52	0.07	0.31	0.68	0.05	0.25	0.63	0.03	0.16
25041	0.08	0.02	0.40	0.15	0.01	0.33	0.05	0.00	0.19	0.00	0.00	0.09
13907	0.01	0.16	0.83	0.07	0.16	0.83	0.23	0.08	0.83	0.00	0.00	0.67
23096	0.04	0.13	0.73	0.29	0.13	0.55	0.50	0.10	0.55	0.50	0.10	0.55
24668	0.00	0.00	0.33	0.00	0.00	0.33	N/A	N/A	0.00	N/A	N/A	0.00
23587	0.29	0.12	0.44	0.22	0.05	0.24	0.21	0.02	0.15	0.50	0.04	0.09
24730	0.00	0.00	0.25	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
16817	0.00	0.00	0.33	0.00	0.00	0.33	N/A	N/A	0.00	N/A	N/A	0.00
24165	0.38	0.10	0.38	0.00	0.00	0.25	0.00	0.00	0.25	0.00	0.00	0.13
24406	0.00	0.00	0.50	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24424	0.00	0.00	0.67	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33
24449	0.00	0.00	0.67	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33
24594	0.19	0.22	0.43	0.36	0.22	0.43	0.03	0.17	0.29	0.08	0.17	0.29
24622	0.00	0.00	0.50	0.00	0.00	0.50	N/A	N/A	0.00	N/A	N/A	0.00
24756	0.06	1.00	1.00	0.13	1.00	1.00	0.27	1.00	1.00	0.09	0.33	1.00
24828	0.00	0.00	0.08	0.00	0.00	0.08	0.00	0.00	0.08	0.00	0.00	0.08
25124	0.98	0.01	0.08	0.99	0.02	0.02	0.99	0.02	0.02	0.99	0.02	0.02
25133	0.31	0.14	0.21	0.31	0.08	0.21	0.14	0.07	0.16	0.38	0.07	0.16
Mean	0.16	0.14	0.49	0.21	0.15	0.37	0.26	0.16	0.28	0.26	0.10	0.25

Bug ID	$\omega_{\text{file}} = 6$			$\omega_{\text{file}} = 7$			$\omega_{\text{file}} = 8$			$\omega_{\text{file}} = 9$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	0.81	0.70	0.93	0.85	0.70	0.90	0.88	0.58	0.90	0.84	0.46	0.66
21330	0.00	0.00	0.05	0.00	0.00	0.05	0.00	0.00	0.05	N/A	N/A	0.00
24657	0.65	0.03	0.16	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25041	0.00	0.00	0.07	0.00	0.00	0.01	N/A	N/A	0.00	N/A	N/A	0.00
13907	0.00	0.00	0.33	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23096	0.75	0.15	0.36	1.00	0.13	0.27	1.00	0.10	0.18	1.00	0.10	0.18
24668	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23587	0.67	0.02	0.09	0.00	0.00	0.03	N/A	N/A	0.00	N/A	N/A	0.00
24730	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
16817	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24165	0.00	0.00	0.13	0.00	0.00	0.13	0.00	0.00	0.13	0.00	0.00	0.13
24406	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24424	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33	N/A	N/A	0.00
24449	0.00	0.00	0.33	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24594	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14	N/A	N/A	0.00
24622	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24756	0.00	0.00	1.00	0.00	0.00	0.67	0.00	0.00	0.33	N/A	N/A	0.00
24828	0.00	0.00	0.08	0.00	0.00	0.08	0.00	0.00	0.08	0.00	0.00	0.08
25124	0.98	0.02	0.02	1.00	0.01	0.02	1.00	0.00	0.01	N/A	N/A	0.00
25133	0.46	0.07	0.16	0.63	0.06	0.11	0.00	0.00	0.05	N/A	N/A	0.00
Mean	0.29	0.07	0.21	0.29	0.07	0.14	0.29	0.07	0.11	0.46	0.14	0.05

Bug ID	$\omega_{\text{file}} = 10$			$\omega_{\text{file}} = 11$			$\omega_{\text{file}} = 12$			$\omega_{\text{file}} = 13$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	0.87	0.48	0.62	0.88	0.45	0.62	0.90	0.37	0.62	0.94	0.27	0.55
21330	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24657	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25041	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
13907	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23096	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24668	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23587	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24730	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
16817	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24165	0.00	0.00	0.13	0.00	0.00	0.13	0.00	0.00	0.13	0.00	0.00	0.13
24406	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24424	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24449	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24594	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24622	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24756	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24828	0.00	0.00	0.08	0.00	0.00	0.08	0.00	0.00	0.08	0.00	0.00	0.04
25124	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25133	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
Mean	0.29	0.16	0.04	0.29	0.15	0.04	0.30	0.12	0.04	0.94	0.27	0.04

Bug ID	$\omega_{\text{file}} = 14$			$\omega_{\text{file}} = 15$			$\omega_{\text{file}} = 16$			$\omega_{\text{file}} = 17$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	1.00	0.16	0.34	1.00	0.17	0.31	1.00	0.15	0.31	1.00	0.13	0.28
21330	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24657	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25041	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
13907	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23096	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24668	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23587	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24730	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
16817	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24165	0.00	0.00	0.13	0.00	0.00	0.13	0.00	0.00	0.13	N/A	N/A	0.00
24406	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24424	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24449	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24594	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24622	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24756	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24828	0.00	0.00	0.04	0.00	0.00	0.04	0.00	0.00	0.04	0.00	0.00	0.04
25124	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25133	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
Mean	0.33	0.05	0.03	0.33	0.06	0.02	0.33	0.05	0.02	0.50	0.07	0.02

Bug ID	$\omega_{\text{file}} = 18$			$\omega_{\text{file}} = 19$			$\omega_{\text{file}} = 20$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	1.00	0.11	0.14	1.00	0.04	0.07	N/A	N/A	0.00
21330	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24657	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25041	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
13907	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23096	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24668	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23587	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24730	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
16817	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24165	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24406	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24424	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24449	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24594	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24622	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24756	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24828	0.00	0.00	0.04	0.00	0.00	0.04	0.00	0.00	0.04
25124	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25133	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
Mean	0.50	0.05	0.01	0.50	0.02	0.01	0.00	0.00	0.00

A.4 Task-wise Results for the File-Level CHB Approach

Bug ID	$\omega_{\text{file}} = 2$			$\omega_{\text{file}} = 3$			$\omega_{\text{file}} = 4$			$\omega_{\text{file}} = 5$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	0.42	0.87	1.00	0.76	0.87	1.00	0.81	0.74	0.97	0.81	0.72	0.93
21330	0.13	1.00	1.00	0.23	0.86	1.00	0.28	0.68	1.00	0.27	0.50	1.00
24657	0.05	0.80	0.83	0.06	0.64	0.83	0.10	0.64	0.83	0.10	0.48	0.83
25041	0.30	0.40	0.96	0.42	0.26	0.96	0.49	0.20	0.93	0.46	0.15	0.82
13907	0.02	0.50	1.00	0.03	0.50	1.00	0.05	0.50	1.00	0.08	0.50	1.00
23096	0.03	0.48	1.00	0.04	0.44	0.86	0.09	0.44	0.86	0.09	0.33	0.86
23587	0.29	0.54	1.00	0.41	0.46	1.00	0.45	0.36	1.00	0.50	0.32	1.00
24730	0.01	0.50	0.67	0.04	0.50	0.67	0.06	0.50	0.67	0.11	0.50	0.67
16817	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00
24165	0.04	0.33	1.00	0.00	0.00	1.00	0.00	0.00	0.67	0.00	0.00	0.67
24406	0.09	0.50	0.67	0.18	0.50	0.67	0.00	0.00	0.33	0.00	0.00	0.33
24424	0.02	0.33	1.00	0.04	0.33	1.00	0.12	0.33	1.00	0.28	0.33	1.00
24594	0.00	0.00	0.50	0.00	0.00	0.50	0.00	0.00	0.50	0.00	0.00	0.50
24828	0.02	0.17	0.29	0.00	0.00	0.29	0.00	0.00	0.29	0.00	0.00	0.29
25124	0.06	0.13	0.80	0.16	0.17	0.60	0.00	0.00	0.40	0.00	0.00	0.40
25133	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17
Mean	0.09	0.41	0.81	0.15	0.35	0.36	0.15	0.27	0.28	0.17	0.24	0.25

Bug ID	$\omega_{\text{file}} = 6$			$\omega_{\text{file}} = 7$			$\omega_{\text{file}} = 8$			$\omega_{\text{file}} = 9$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	0.86	0.67	0.93	0.84	0.45	0.69	0.87	0.46	0.66	0.88	0.48	0.62
21330	0.33	0.41	0.88	0.35	0.29	0.88	0.36	0.24	0.88	0.36	0.19	0.75
24657	0.13	0.40	0.83	0.12	0.32	0.83	0.13	0.24	0.83	0.15	0.16	0.83
25041	0.47	0.14	0.61	0.46	0.12	0.57	0.38	0.09	0.54	0.44	0.07	0.46
13907	0.09	0.33	1.00	0.11	0.33	1.00	0.13	0.33	0.50	0.17	0.33	0.50
23096	0.10	0.20	0.71	0.13	0.20	0.71	0.20	0.17	0.57	0.42	0.17	0.57
23587	0.43	0.33	0.75	0.61	0.40	0.63	0.62	0.34	0.63	0.78	0.43	0.50
24730	0.15	0.50	0.67	0.56	0.50	0.67	0.56	0.50	0.67	0.56	0.50	0.67
16817	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.50	0.00	0.00	0.50
24165	0.00	0.00	0.67	0.00	0.00	0.67	0.00	0.00	0.33	0.00	0.00	0.33
24406	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33
24424	0.33	0.33	1.00	0.33	0.33	1.00	0.50	0.50	0.67	0.00	0.00	0.67
24594	0.00	0.00	0.50	0.00	0.00	0.50	0.00	0.00	0.50	0.00	0.00	0.50
24828	0.00	0.00	0.29	0.00	0.00	0.29	0.00	0.00	0.29	0.00	0.00	0.29
25124	0.00	0.00	0.40	0.00	0.00	0.40	0.00	0.00	0.40	0.00	0.00	0.20
25133	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17
Mean	0.18	0.21	0.22	0.22	0.18	0.19	0.24	0.18	0.19	0.23	0.15	0.16

Bug ID	$\omega_{\text{file}} = 10$			$\omega_{\text{file}} = 11$			$\omega_{\text{file}} = 12$			$\omega_{\text{file}} = 13$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	0.88	0.41	0.62	0.92	0.34	0.59	0.91	0.15	0.48	0.95	0.14	0.38
21330	0.33	0.14	0.75	0.37	0.14	0.75	0.35	0.14	0.50	0.54	0.14	0.50
24657	0.17	0.16	0.83	0.29	0.10	0.67	0.30	0.10	0.67	0.44	0.13	0.50
25041	0.46	0.07	0.43	0.45	0.09	0.29	0.38	0.08	0.25	0.41	0.08	0.25
13907	0.31	0.33	0.50	0.56	0.33	0.50	0.58	0.33	0.50	0.63	0.33	0.50
23096	0.21	0.11	0.43	0.38	0.11	0.43	0.39	0.11	0.43	0.42	0.11	0.43
23587	0.86	0.43	0.50	0.86	0.36	0.50	0.83	0.29	0.50	0.80	0.29	0.38
24730	0.60	0.50	0.67	0.60	0.50	0.67	0.00	0.00	0.33	0.00	0.00	0.33
16817	0.00	0.00	0.50	0.00	0.00	0.50	0.00	0.00	0.50	0.00	0.00	0.50
24165	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33
24406	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33
24424	0.00	0.00	0.67	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24594	0.00	0.00	0.50	0.00	0.00	0.50	0.00	0.00	0.50	N/A	N/A	0.00
24828	0.00	0.00	0.29	0.00	0.00	0.29	0.00	0.00	0.29	0.00	0.00	0.14
25124	0.00	0.00	0.20	0.00	0.00	0.20	0.00	0.00	0.20	0.00	0.00	0.20
25133	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	N/A	N/A	0.00
Mean	0.24	0.14	0.15	0.29	0.13	0.42	0.25	0.08	0.37	0.32	0.09	0.30

Bug ID	$\omega_{\text{file}} = 14$			$\omega_{\text{file}} = 15$			$\omega_{\text{file}} = 16$			$\omega_{\text{file}} = 17$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	0.94	0.15	0.31	1.00	0.13	0.28	1.00	0.08	0.21	1.00	0.04	0.14
21330	0.00	0.00	0.25	0.00	0.00	0.25	0.00	0.00	0.25	0.00	0.00	0.25
24657	0.44	0.13	0.50	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33
25041	0.41	0.06	0.25	0.45	0.05	0.25	0.47	0.05	0.25	0.52	0.04	0.25
13907	0.75	0.33	0.50	1.00	0.33	0.50	1.00	0.33	0.50	1.00	0.33	0.50
23096	0.50	0.11	0.43	0.67	0.11	0.43	0.67	0.11	0.43	0.67	0.11	0.43
23587	1.00	0.19	0.38	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24730	0.00	0.00	0.33	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
16817	0.00	0.00	0.50	0.00	0.00	0.50	0.00	0.00	0.50	N/A	N/A	0.00
24165	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33	N/A	N/A	0.00
24406	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33
24424	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24594	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24828	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14
25124	0.00	0.00	0.20	0.00	0.00	0.20	0.00	0.00	0.20	0.00	0.00	0.20
25133	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
Mean	0.31	0.08	0.28	0.28	0.06	0.22	0.29	0.05	0.22	0.35	0.06	0.16

Bug ID	$\omega_{\text{file}} = 18$			$\omega_{\text{file}} = 19$			$\omega_{\text{file}} = 20$			$\omega_{\text{file}} = 21$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	1.00	0.04	0.07	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
21330	0.00	0.00	0.13	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24657	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.17	0.00	0.00	0.17
25041	0.52	0.04	0.25	0.43	0.04	0.21	0.44	0.04	0.21	0.29	0.03	0.18
13907	1.00	0.33	0.50	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23096	0.67	0.11	0.43	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14
23587	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24730	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
16817	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24165	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24406	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33	0.00	0.00	0.33
24424	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24594	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24828	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14
25124	0.00	0.00	0.20	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25133	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
Mean	0.35	0.06	0.15	0.09	0.01	0.07	0.09	0.01	0.06	0.06	0.01	0.06

Bug ID	$\omega_{\text{file}} = 22$			$\omega_{\text{file}} = 23$			$\omega_{\text{file}} = 24$			$\omega_{\text{file}} = 25$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
21330	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24657	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17
25041	0.49	0.03	0.18	0.49	0.03	0.18	0.49	0.03	0.18	0.51	0.03	0.18
13907	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23096	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14	N/A	N/A	0.00
23587	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24730	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
16817	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24165	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24406	0.00	0.00	0.33	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24424	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24594	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24828	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14
25124	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25133	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
Mean	0.10	0.01	0.06	0.12	0.01	0.04	0.12	0.01	0.04	0.17	0.01	0.03

Bug ID	$\omega_{\text{file}} = 26$			$\omega_{\text{file}} = 27$			$\omega_{\text{file}} = 28$			$\omega_{\text{file}} = 29$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
21330	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24657	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17	0.00	0.00	0.17
25041	0.25	0.02	0.11	0.25	0.02	0.11	0.38	0.04	0.07	0.38	0.04	0.07
13907	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23096	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
23587	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24730	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
16817	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24165	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24406	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24424	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24594	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
24828	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.14
25124	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
25133	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00	N/A	N/A	0.00
Mean	0.08	0.01	0.03	0.08	0.01	0.03	0.13	0.01	0.02	0.13	0.01	0.02

Bug ID	$\omega_{\text{file}} = 30$		
	$\widetilde{\text{pr}_m}$	$\widetilde{\text{rc}_m}$	$\widetilde{\text{tp}_m}$
24635	N/A	N/A	0.00
21330	N/A	N/A	0.00
24657	0.00	0.00	0.17
25041	0.38	0.04	0.07
13907	N/A	N/A	0.00
23096	N/A	N/A	0.00
23587	N/A	N/A	0.00
24730	N/A	N/A	0.00
16817	N/A	N/A	0.00
24165	N/A	N/A	0.00
24406	N/A	N/A	0.00
24424	N/A	N/A	0.00
24594	N/A	N/A	0.00
24828	0.00	0.00	0.14
25124	N/A	N/A	0.00
25133	N/A	N/A	0.00
Mean	0.13	0.01	0.02

A.5 Additional Results for the Method-Level CHB Approach

	CHB, file-level			CHB, method-level		
ω	$\mathrm{pr}'_{\mathrm{avg}}$	$\mathrm{rc}'_{\mathrm{avg}}$	$\mathrm{tp}'_{\mathrm{avg}}$	$\mathrm{pr}'_{\mathrm{avg}}$	$\mathrm{rc}'_{\mathrm{avg}}$	$\mathrm{tp}'_{\mathrm{avg}}$
2	0.23	0.58	0.87	0.36	0.20	0.23
3	0.38	0.51	0.85	0.35	0.28	0.15
4	0.43	0.44	0.81	0.49	0.31	0.12
5	0.42	0.39	0.78	0.53	0.29	0.11

Bibliography

[1] Eclipse build notes for latest release 2.0. `http://archive.eclipse.org/eclipse/downloads/drops/R-2.0-200206271835/buildNotes.php`.

[2] Eclipse build notes for latest release 2.1. `http://archive.eclipse.org/eclipse/downloads/drops/R-2.1-200303272130/buildNotes.php`.

[3] Deepak Advani, Youssef Hassoun, and Steve Counsell. Extracting refactoring trends from open-source software and a possible solution to the 'related refactoring' conundrum. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 1713–1720, 2006.

[4] Hiralal Agrawal and Joseph R. Horgan. Dynamic program slicing. In *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation*, pages 246–256, 1990.

[5] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[6] Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold. A differencing algorithm for object-oriented programs. In *Proceedings of the 19th IEEE International Conference on Automated Software Engineering*, pages 2–13, 2004.

[7] Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold. Efficient and precise dynamic impact analysis using execute-after sequences. In *Proceedings of the 27th International Conference on Software Engineering*, pages 432–441, 2005.

[8] Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold. Jdiff: A differencing technique and tool for object-oriented programs. *Automated Software Engineering*,

14(1):3–36, 2007.

[9] L. A. Belady and M. M. Lehman. A model of large program development. *IBM Systems Journal*, 15(3):225–252, 1976.

[10] T. J. Biggerstaff. The library scaling problem and the limits of concrete component reuse. In *Proceedings of the 3rd International Conference on Software Reuse*, pages 102–109, 1994.

[11] Elizabeth Burd and Malcolm Munro. Investigating the maintenance implications of the replication of code. In *Proceedings of the International Conference on Software Maintenance*, page 322, 1997.

[12] Doug Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 443–452, 2001.

[13] Gerardo Canfora and Luigi Cerulo. Impact analysis by mining software and change request repositories. In *Proceedings of the 11th IEEE International Software Metrics Symposium*, pages 29–1–29–9, 2005.

[14] Gerardo Canfora and Luigi Cerulo. Fine grained indexing of software repositories to support impact analysis. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 105–111, 2006.

[15] Davor Čubranić and Gail C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering*, pages 408–418, 2003.

[16] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *Proceedings of the 21st international conference on Software engineering*, pages 285–294, 1999.

[17] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 1999.

[18] Harald Gall, Karin Hajek, and Mehdi Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance*, pages 190–198, 1998.

[19] Karam Gouda and Mohammed J. Zaki. Efficiently mining maximal frequent itemsets. In *Proceedings of the IEEE International Conference on Data Mining*, pages 163–170, 2001.

[20] Gösta Grahne and Jianfei Zhu. High performance mining of maximal frequent itemsets. In *Proceedings of the SIAM '03 Workshop on High Performance Data Mining: Pervasive and Data Stream Mining*, pages 135–143, 2003.

[21] William G. Griswold and David Notkin. Automated assistance for program restructuring. *ACM Transactions on Software Engineering and Methodology*, 2(3):228–269, 1993.

[22] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.

[23] Reid Holmes and Gail C. Murphy. Using structural context to recommend source code examples. In *Proceedings of the 27th International Conference on Software Engineering*, pages 117–125, 2005.

[24] Reid Holmes and Robert J. Walker. Supporting the investigation and planning of pragmatic reuse tasks. In *Proceedings of the International Conference on Software Engineering*, pages 447–457, 2007.

[25] Miryung Kim and David Notkin. Using a clone genealogy extractor for understanding

and supporting evolution of code clones. In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, pages 1–5, 2005.

[26] Miryung Kim, David Notkin, and Dan Grossman. Automatic inference of structural changes for matching across program versions. In *Proceedings of the 29th International Conference on Software Engineering*, pages 333–343, 2007.

[27] Miryung Kim, Vibha Sazawal, David Notkin, and Gail Murphy. An empirical study of code clone genealogies. In *Proceedings of the 10th European Software Engineering Conference held jointly with the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 187–196, 2005.

[28] Sunghun Kim, Kai Pan, and E. James Whitehead, Jr. When functions change their names: Automatic detection of origin relationships. In *Proceedings of the 12th Working Conference on Reverse Engineering*, pages 143–152, 2005.

[29] K. Kontogiannis. Evaluation experiments on the detection of programming patterns using software metrics. In *Proceedings of the Fourth Working Conference on Reverse Engineering*, pages 44–54, 1997.

[30] J. Laski and W. Szermer. Identification of program modifications and its applications in software maintenance. In *Proceedings of the 1992 Conference on Software Maintenance*, 1992.

[31] Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering*, pages 492–501, 2006.

[32] M. M. Lehman and F. N. Parr. Program evolution and its impact on software engineering. In *Proceedings of the 2nd International Conference on Software Engineering*, pages 350–357, 1976.

[33] B.P. Lientz and E.B. Swanson. *Software Maintenance Management*. Addison-Wesley, 1980.

[34] Benjamin Livshits and Thomas Zimmermann. DynaMine: Finding common error patterns by mining software revision histories. In *Proceedings of the 10th European Software Engineering Conference held jointly with the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 296–305, 2005.

[35] Audris Mockus, Roy Fielding, and James Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[36] Alessandro Orso, Nanjuan Shi, and Mary Jean Harrold. Scaling regression testing to large software systems. In *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 241–251, 2004.

[37] Janice Singer. Practices of software maintenance. In *Proceedings of the International Conference on Software Maintenance*, page 139, 1998.

[38] Robert J. Walker, Reid Holmes, Ian Hedgeland, Puneet Kapur, and Andrew Smith. A lightweight approach to technical risk estimation via probabilistic impact analysis. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 98–104, 2006.

[39] Mark Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, pages 439–449, 1981.

[40] Zhenchang Xing and Eleni Stroulia. Refactoring practice: How it is and how it should be supported - an eclipse case study. In *ICSM '06: Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 458–468, 2006.

[41] Annie T. T. Ying, Gail C. Murphy, Raymond Ng, and Mark C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9):574–586, 2004.

[42] Annie Tsui Tsui Ying. Predicting software changes by mining revision history. Master of Science thesis, Department of Computer Science, University of British Columbia, 2003.

[43] Liguo Yu and Srini Ramaswamy. Change propagations in the maintenance of kernel-based software with a study on linux. In *Proceedings of the 45th Annual Southeast Regional Conference*, pages 76–81, 2007.

[44] Thomas Zimmermann, Peter Weissgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005.

[45] Lijie Zou and Michael W. Godfrey. Using origin analysis to detect merging and splitting of source code entities. *IEEE Transactions on Software Engineering*, 31(2):166–181, 2005.