SRDIFF: SYNTACTIC DIFFERING TO SUPPORT SOFTWARE MAINTENANCE AND EVOLUTION

A dissertation submitted
to Kent State University in partial
fulfillment of the requirements for the
degree of Doctor of Philosophy

by
Michael John Decker

August 19, 2017
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................ VII

LIST OF TABLES ........................................................................................................... XII

DEDICATION .................................................................................................................. XVIII

ACKNOWLEDGEMENTS ............................................................................................... XIX

CHAPTER 1 INTRODUCTION ....................................................................................... 1
  1.1 Research Questions ............................................................................................... 6
  1.2 Research Contributions ......................................................................................... 7
  1.3 Organization ........................................................................................................... 8

CHAPTER 2 BACKGROUND AND RELATED WORK ............................................... 9
  2.1 Textual Differencing ............................................................................................ 10
  2.2 Syntactic Differencing ......................................................................................... 13
  2.3 Semantic Differencing ......................................................................................... 16
  2.4 Track-changes Approaches .................................................................................. 17
  2.5 File Comparison Software ................................................................................... 17
  2.6 Other Differencing Approaches ......................................................................... 18
  2.7 Version Control Systems ..................................................................................... 18
  2.8 Software Merging ................................................................................................ 19
  2.9 Source-code Representations .............................................................................. 22
  2.10 Improving Upon the Literature ......................................................................... 22

CHAPTER 3 SRCML INFRASTRUCTURE .................................................................. 25
  3.1 srcML Format ...................................................................................................... 26
CHAPTER 7 EVALUATION ................................................................. 86
7.1 Data Collection and Preparation .................................................. 89
7.2 Measures ............................................................................. 91
7.3 Results and Discussion ............................................................. 92
7.4 Timing ................................................................................. 96
7.5 Threats to Validity ................................................................. 97

CHAPTER 8 INVESTIGATING METHOD STEREOTYPE HISTORY ........ 100
8.1 Method Stereotypes ............................................................... 100
8.2 Evolution of a Method’s Stereotype .......................................... 103
8.3 Data Collection .................................................................... 106
8.4 Results and Discussion ........................................................... 111
  8.4.1 Empirical Results ............................................................. 111
  8.4.2 Manual Investigation ......................................................... 117
8.5 Threats to Validity ................................................................. 120
8.6 Conclusion ............................................................................. 123

CHAPTER 9 INVESTIGATING CODE REWRITE/REPLACEMENT AND CODE
CONVERSION .................................................................................... 124
9.1 Rewrite and Replacement ........................................................ 125
9.2 Conversion ............................................................................ 127
9.3 Data Collection ...................................................................... 130
9.4 Results and Discussion ........................................................... 135
  9.4.1 Replace Results and Discussion .......................................... 136
9.4.2 Convert Results and Discussion................................................................. 146

9.5 Threats to Validity.......................................................................................... 152

9.6 Conclusion..................................................................................................... 154

CHAPTER 10 CONCLUSION.................................................................................. 155

APPENDIX A CODE FOR SRCDIFF TOOL INTEGRATION INTO GIT........ 158

APPENDIX B COMPLETE SRCDIFF TOOL OPTIONS ..................................... 159

APPENDIX C SOURCE-CODE CHANGES USED IN PRELIMINARY STUDY 161

APPENDIX D SOURCE-CODE CHANGES USED IN THE EVALUATION...... 191

REFERENCES...................................................................................................... 302
LIST OF FIGURES

Figure 1.1. Example change showing the deficiency of textual differencing approaches on source code. The top-left is the original source code. A \{ \ was added at the end of the first line, and a new line with a \} was added at the end, resulting in the code on the top-right. The resulting textual difference (line difference) is shown on the bottom. A deleted line is shown with a preceding – and an insert with a preceding +. Background colors have been added to highlight the change with red for delete and green for insert. There are two problems with the line difference. First, the addition of the \{ \ is much smaller than a line, yet the whole line is marked as deleted and inserted. Secondly, the logical change is the addition of the block as a whole \{ \} where several separate modifications are indicated here.

Figure 1.2. A simple example of a line-diff where two associated lines are not logically grouped together. In this example, the method call to \texttt{anOrder.basePrice()} is extracted and replaced with a new variable \texttt{basePrice}. Even though, it is the same return statement, another edit lies between the inserted and deleted lines.

Figure 2.1. The output of GNU diff on a method \texttt{setImage} (simplified to fit) from KOffice revisions 1026809-1026810. On the top is the original and modified function. The middle is the side-by-side view, and the bottom is the unified view.

Figure 3.1. Diagram of the srcML Infrastructure. The foundation is the srcML Format. On top of this is the srcML tool, which converts source code to and from the srcML Format. From this core functionality, XML technologies and additional models can
be leveraged to develop tools to support exploration, analysis, and manipulation of large software systems. The infrastructure is supported (right) via a multi-university development team. 26

Figure 3.2. Example source code function and its markup in srcML. Syntax information such as a section of code being a function or an expression statement is encoded by wrapping the appropriate source code with XML tags. The process is lossless. All the original text, including whitespace and comments is preserved. 27

Figure 3.3. Example srcML archive. A separate unit tag is added as a root tag around the file unit tags. 28

Figure 4.1. Example source-code change. The top-left contains the original source code and the top-right contains the modified source code. The original and modified code contained within the srcDiff format is given. Deletions are highlighted in red and with a line-thought mark. Insertions are highlighted with green. All the original source code is placed in bold. 36

Figure 4.2. An example of the changed methods from Figure 4.1 in two different human readable formats produced by srcDiff. The top is a side-by-side view of the change, where the original method is on the left with deleted code marked with a red background and the modified method is on the right with inserted code marked with a green background. The bottom is a unified view where both methods are merged into one with deleted code marked with a red background and inserted code marked with a green background. 37
Figure 4.3. The srcDiff algorithm. A preorder traversal is performed on both the original and modified trees. The input is the original and modified children of a node. A sequence-differencing method is used to categorize the children as common, unique to one version, or changed. Common, inserted, and deleted children are output directly. Changed children are further analyzed in determineMatchings and determineNestings for the following situations: a child in the original is a previous version of a child in the modified, a child and its subtree in one version subsumes children in the other version, and a child is unique to one version. The functions determineMatchings and determineNestings use a set of rules to identify these situations ................................................. 42

Figure 4.4. Example of the srcDiff process. a) The original and modified version of a declaration statement with their tree representations below each. b) With decl_stmt as common root, run a sequence diff on their child sequences (blue). Compares are between entire child subtrees. c) Output of sequential diff. red is delete edit, green is insert edit, remaining are common. <specifier> subtree is an insert and <init>’s are a change. Common and insert/delete require no further action. The change needs to be analyzed to see if Match, Nested, or Inserted/Deleted ........................................................................................................ 43

Figure 4.5. Textual similarity rules. The rules use similarity, dissimilarity and size measures to test if two elements are similar enough textually. The measures, with exception to rule 2, are calculated after stripping out punctuation characters and
other tokens that negatively affect performance. Rule 2 uses all tokens to test for an exact match. It is not applied to *Nesting* rules.  

Figure 5.1. A simplified high-level view of modules in `srcdiff`.  

Figure 5.2. Examples of `srcDiff` output provided by the `view` module. Top shows the original and modified source code, middle is the default theme and default level of syntax highlighting. Bottom is the Monokai Theme with full syntax highlighting. Both outputs are taken from `srcDiff`’s `view` module to HTML.  

Figure 6.1. Answers to preferability question (Q5) as a stacked bar chart. Blue is the percentage selected as most preferable, green second preferable, and red least preferable.  

Figure 9.1. Example code replacement. The body of the for-statement is completely deleted and replaced with a different body.  

Figure 9.2. Simplified example of `srcDiff` markup of code replacement. Red is delete and green is insert. A `diff:delete` tag and `diff:insert` tag are issued consecutively (one after the other) and they both are given the type `replace`. Here an expression-statement, if-statement, and while-statement are replaced with a declaration-statement and a different expression-statement.  

Figure 9.3. Example code conversion. The original code is on the top-left, modified top-right, and `srcDiff` on the bottom. In the example, the for-statement was converted into an equivalent while-statement. The body of the for-statement is largely unchanged.
Figure 9.4. Output of GumTree on a simple code conversion. Red is delete, yellow-green insert, and blue is move. A while-statement is converted to a for-statement.

GumTree marks the while-statement deleted and for-statement inserted and a series of moves between the two.

Figure 9.5. Example markup of a code conversion in srcDiff. Red is delete and green is insert. A \texttt{diff:delete} is wrapped around the original syntactic element (and keyword if present) with the type \texttt{convert}, and a \texttt{diff:insert} is wrapped around the modified syntactic element (and keyword if present) with type \texttt{convert}.

The contents are then marked as common, inserted, and deleted as necessary.

Figure 9.6. The total number of revisions plotted against the number of replaces. X-axis is number of revisions and y-axis is the number of replaces. The result is fairly linear.

Figure 9.7. The total number of revisions plotted against the number of conversions. X-axis is number of revisions and y-axis is the number of conversions. The result is fairly linear.
LIST OF TABLES

Table 4.1. Match rules and the associated syntactic categories (srcML elements). ........ 48

Table 4.2. Convertibility Rules organized by type. Each type has the associated syntactic categories........................................................................................................... 56

Table 6.1. List of null and alternate hypothesis tested for questions: Q1-Q4................. 78

Table 6.2. Results of Two-tailed Paired T-Tests on Q1 and Q2. Left-most column is the sample, and top-most is the questions. For each question, the mean of GNU diff and srcDiff views are given along with the t-test p-value and finally if we can reject the null hypothesis .................................................................................................................. 79

Table 6.3. Results of Two-tailed Paired T-Tests on Q3-Q4. Left-most column is the sample, and top-most is the questions. For each question, the mean of GNU diff and srcDiff views are given along with the t-test p-value and finally if we can reject the null hypothesis .................................................................................................................. 80

Table 6.4. Effect size (Cohen’s d) for each sample and for Q1 and Q2. In general, there was a small effect for Accuracy Changed (Q1) and almost no effect for Accuracy Common (Q2)........................................................................................................................................ 80

Table 6.5. Effect size (Cohen’s d) for each sample and for Q3 and Q4. In general, there was a medium to large effect on Accuracy (Q3) and Understandability (Q4). ........ 81

Table 7.1. Evaluation survey questions. .................................................................................................................. 92
Table 7.2. Results for each of the samples and post-questionnaire (Overall). All reported measures are medians with GitHub (GH), Mergerly (M), srcDiff (SD), and GumTree (GT). Last line shows a count of the number of samples (and Overall) for which the approaches achieved the best result (including ties) out of all the approaches. For Changed, smaller number is better, others larger is better.

Table 7.3. Results for each of the samples and post-questionnaire (Overall). All reported measures are medians with GitHub (GH), Mergerly (M), srcDiff (SD), and GumTree (GT). Last line shows a count of the number of samples (and Overall) for which the approaches achieved the best result (including ties) out of all the approaches. Larger is better.

Table 8.1. Method stereotype taxonomy. The far-left column is the broad stereotype categories, the middle column the individual stereotypes, and the right column is a description of each stereotype.

Table 8.2. Categories of important stereotype transitions. The category-column lists the categories of stereotype changes. The group-column groups similar categories, the description-column is a description of the category and what belong as part of that category, and the notes-column contains notes about the transitions.

Table 8.3. Method stereotype transition categories (continued).

Table 8.4. List of selected software systems by alphabetical order. Left column is system, middle is number of revisions with modified C++ files, and right is the repository location. Unqualified locations are from GitHub: https://github.com/ {Location}.git.
Table 8.5. Selected systems (continued).................................................................................. 108

Table 8.6. Top ten most common stereotype transitions. from-column is original version stereotype, to-column is the modified version stereotype, count is the number of times it occurred, and % is the percentage of occurrence out of the all transitions. 112

Table 8.7. Top ten stereotype type transitions where the stereotype changed. from-column is the original version stereotype, to-column is modified version stereotype, count-column is the number of occurrences, and %-column is the percentage it occurred out of all transitions. ................................................................................................................. 113

Table 8.8. Top ten purely deleted stereotypes. Left-column is the stereotypes deleted, the middle-column is the number of times those stereotypes were the only change in method stereotype, and the right-column is the percent out of all method stereotypes that changed. ................................................................................................................. 114

Table 8.9. Top ten purely inserted stereotypes. Left-column is the stereotypes inserted, the middle-column is the number of times those stereotypes were the only change in method stereotype, and the right-column is the percent out of all method stereotypes that changed. ................................................................................................................. 115

Table 8.10. Top ten completely changed stereotypes. The from-column is the original version stereotype, the to-column is the modified version stereotype, count-column is the number of occurrences and %-column is the percentage of occurrence out of all method stereotypes that changed. ................................................................................................................. 116

Table 8.11. Top ten replaced stereotypes. The from-column is what was replaced, the to-column what it was replaced with, count-column is the number of occurrences, and
%-column is the percentage of occurrence out of all method stereotypes that changed. ................................................................. 116

Table 8.12. Summary of manual investigation. The category-column contains method stereotype change categories. Count is the number of times that category appeared. The suspicious-column contains the count and percent of the number of times that category had a change that should be investigated more deeply, and the benign-column is the count and percentage of the times that category was of no concern. 119

Table 9.1. Code conversion supported by srcDiff. Left is category and right are the syntactic elements that can be converted to each other. ............................... 130

Table 9.2. List of selected software systems by alphabetical order. Left column is system, middle is number of revisions with modified C++ files, and right is the repository location. Unqualified locations are from GitHub:

https://github.com/{Location}.git......................................................... 132

Table 9.3. Selected systems (continued)...................................................... 133

Table 9.4. Number of replaces per system (alphabetically). First column is system, the second is the number of replaces, and the average per revision. .......................... 137

Table 9.5. Number of replaces (continued). .................................................. 138

Table 9.6. Deleted elements in a replace. Left column is what element is deleted, the middle column is how many instances were deleted in all replaces, and the third column is the number each is replaced per revision. ................................. 140
Table 9.7. Inserted elements in a replace. Left column is what element is deleted, the middle column is how many instances were deleted in all replaces, and the third column is the number each is replaced per revision........................................ 141

Table 9.8. Number times an element was replaced with itself. The left column is the element, the middle column is number of occurrence, and the right column is the number (on average) that occurred in each revision........................................ 142

Table 9.9. Normalized deleted elements in a replace. The left-column is the deleted element. The right-column (normalized per revision) is calculated as: for each system, each element is normalized by the number of revisions for that system and the median across all systems is taken........................................ 143

Table 9.10. Normalized inserted elements in a replace. The left-column is the inserted element. The right-column (normalized per revision) is calculated as: for each system, each element is normalized by the number of revisions for that system and the median across all systems is taken........................................ 144

Table 9.11. Normalized times an element was replaced with itself. The left-column is the element. The right-column (normalized per revision) is calculated as: for each system, each element is normalized by the number of revisions for that system and the median across all systems is taken........................................ 145

Table 9.12. Total number of conversions for each system across all revisions. Left-column is system, middle is the number of converts, and the right column is the number of converts divided by revision (as percent). Last row contains total overall systems. ......................................................... 148
Table 9.13. Total conversions (continued). ................................................................. 149

Table 9.14. Frequency of each time of conversion. The left column is the number of conversions, the middle column is the raw number of how many times the conversion occurred between all systems, and the right column is the percentage that conversion occurred out of all types. ................................................................. 150

Table 9.15. Percentage of each type of conversion normalized for each system by the total number of conversions. ............................................................................................... 151
DEDICATION

To the one True God, creator of Heaven and of Earth.

Soli Deo gloria

To my mother, sister, father, and the rest of my family who cared for and supported me in so many ways throughout my life.
ACKNOWLEDGEMENTS

I owe a great many people for helping me through this dissertation. I acknowledge only a few of them here. First, my Master’s advisor, Michael L. Collard, and my Ph.D. advisor, Jonathan I. Maletic, who I owe very much for molding me into the researcher I am now. For the people who helped with various parts of my research such as Heather M. Guarnera who double checked my algorithm complexity, Natalia Dragan who identified the interesting method-stereotype changes, and Christian D. Newman who helped with confirming the method-stereotype manual analysis. I am also indebted to the members of the Software Development Laboratory (SDML) for their friendship, advice, and everything else. All of you are very dear to me. I hope to continue working with all of you long into the future.

I also would like to thank my family, friends, and members of Zion Evangelical Lutheran Church in Valley City for supporting me, as well as, my cat, Shampoo, for help keeping me sane.

Michael John Decker
July 10, 2017, Kent, Ohio
CHAPTER 1

Introduction

Developing and maintaining software is an iterative process of change. That is, software developers constantly improve and modify (add to and delete from) a code base a little at a time. In a system modified by a solitary developer, understanding these changes is generally straightforward. However, when scaled to real-world, large-scale systems (>500,000 lines of code) that involve large teams of individuals located in different geographical locations, understanding what exactly changed in a code base is often problematic and requires substantial effort by developers. It is essential for developers to quickly and easily comprehend changes made to source code. This is critical when understanding the impact of changes, merging changes from multiple developers, conducting code reviews and reviews of commit requests, and tracking down errors arising from changes.

Textual differencing tools such as GNU diff have been in wide use by developers for many years. They provide an extremely efficient means to determine the difference between two versions of a file (or set of files). These tools typically take a line-based view and produce the set of lines that have been inserted and/or deleted between the two versions (i.e., a delta). Textual differencing (diff) is the de facto standard for computing and presenting changes of source code. It is widely incorporated into Integrated Development Environments (e.g., Microsoft Visual Studio, Apple’s Xcode) and version-control systems.
Line-based differencing approaches are very flexible (i.e., can be applied to any text file). They are also highly efficient (i.e., scale well to large files/projects). One of the best known textual differencing algorithm’s (e.g., Myer’s algorithm [Myers 1986]) has a run time complexity of O(ND) where N is the sum of the lengths of both files (i.e., the number of lines) and D is the total number of edits (i.e., number of inserted/deleted lines). When the file is not greatly modified between commits (as is normal with commits), this, in practice, approaches a linear time algorithm. The efficiency and flexibility of textual differencing comes at a price. As they only treat the files as text, they understand nothing about the structure or syntax of the source code being differenced. As such, it is often difficult to understand the output in the context of the language syntax and structure.

In practice, source-code changes are typically more fine-grained than a line (e.g., an identifier is changed) or cross multiple lines (e.g., an entire loop or block is deleted). Moreover, a single logical modification may span multiple non-contiguous lines, such as adding a block (i.e., { }) to an else-clause that previously did not have one. Figure 1.1 shows an example. The top-left shows an if-statement that was modified by adding a { at the end of the first line, and a } in a new line at the end. The top-right is the resulting source code. The bottom shows the resulting textual (line) difference. Deleted lines are preceded with a − and inserted lines with a +. Background colors have been added with red for delete and green for insert. There are two problems with the markup. First, the addition of the { is much smaller than a line, yet as the textual difference can only report line differences, the whole line is marked as deleted and inserted. Secondly, one logical
operation was performed on the code, that is, a block \{ \} was inserted. However, the line difference shows several separate (seemingly unrelated) deletes and inserts that the developer has to cognitively associate.

Another problematic issue is that modifications to a line are represented as a deletion and then an insertion. The related \(-\)delete and \(+\)insert can occur many lines apart in the delta, making it very difficult to understand the (simple) change. Figure 1.2 shows a simple example. A method call to basePrice is extracted out and replaced with a newly created variable. However, the two logically related return-statements are separated with another edit. Again, the developer has to expand further cognitive effort to associate the two.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
</table>
| if(itr != list.end())
list.remove(itr); | if(itr != list.end()) {
             list.remove(itr);
} |

<table>
<thead>
<tr>
<th>Textual Difference</th>
</tr>
</thead>
</table>
| -if(itr != list.end())
+if(itr != list.end()) { |
    list.remove(itr); |
+}

Figure 1.1. Example change showing the deficiency of textual differencing approaches on source code. The top-left is the original source code. A \{ \} was added at the end of the first line, and a new line with a \} was added at the end, resulting in the code on the top-right. The resulting textual difference (line difference) is shown on the bottom. A deleted line is shown with a preceding \(-\) and an insert with a preceding \(+\). Background colors have been added to highlight the change with red for delete and green for insert. There are two problems with the line difference. First, the addition of the \{ \} is much smaller than a line, yet the whole line is marked as deleted and inserted. Secondly, the logical change is the addition of the block as a whole \{ \} where several separate modifications are indicated here.
In short, the line-based view does not respect syntactic boundaries. Thus, the deltas often do not sufficiently or clearly reflect the actual meaning of the change.

The alternative to line-based approaches is syntactic differencing [Apiwattanapong, Orso, Harrold 2007; Fluri, Wursch, Pinzger, Gall 2007; Raghavan, Rohana, Podgurski, Augustine 2004]. Syntactic differencing requires some type comparison on the Abstract Syntax Tree (AST) of the source code. However, these approaches typically ignore changes to whitespace, comments, and preprocessor statements (as they are not in the AST or outside of the formal language syntax). Also, many syntactic differencing tools focus on optimal differences (i.e., shortest edit script) that may not produce a very understandable delta. This can lead to trying to map parts of a version together when in all actuality; a complete replacement of code occurred. Lastly, while syntactic information is very useful, it is not sufficient for generating easy-to-read and understandable deltas. For that, additional rules and idioms about how programmers edit and change code are needed. Moreover, tree-differencing approaches are generally slow (compared to diff) and thus do not scale well to large systems. These are some of the underlying reasons for why tree-
differencing tools have not been widely accepted by developers. More about syntactic differencing will be discussed in CHAPTER 4 along with the approach proposed in this dissertation.

Another approach is to save each keystroke and edit action of the developer(s) while the change is taking place, such as track-changes in a word processor. This information along with syntactic knowledge can produce a delta that reflects the actual syntactic change that occurred. However, this information is generally not available unless a specialized environment is used [Robbes, Lanza 2008]. Furthermore, this information is completely unavailable in the case of two non-consecutive versions of a file or in the case of multiple versions from different sources (multiple developers editing the same file).

The goal of the work presented here is to produce a more understandable and readable delta than what is produced by current line-based and syntactic-differencing approaches. My overarching hypothesis is that on its own, shortest edit script is a poor heuristic for syntactic differencing if the goal is to generate understandable deltas. To investigate this, we developed the tool srcDiff. It produces differences in the srcDiff format [Maletic, Collard 2004], which is an extension to the srcML (www.srcML.org) format [Collard, Decker, Maletic 2013]. srcDiff is a syntactic-differencing approach that works on the AST of source code as represented by srcML. Thus, it respects syntactic boundaries and supports both coarse and fine-grained changes. Unlike pure syntactic-differencing approaches, srcDiff does not use a standard tree-based differencing approach. Instead, it is a hybrid approach that utilizes more efficient sequence differencing, as provided by Myers’ O(ND) Shortest Edit Script algorithm [Myers 1986], to map changes onto the tree
structure. Thus, it combines the efficiency of line differencing with localized tree comparison.

In order to produce a more understandable delta, we use a set of empirically derived rules for selecting the most appropriate syntactic category (granularity) for a given change and to determine the scope of a set of changes within the code. The rules model how programmers make changes to code thus the resulting deltas better reflect their expectations in the context of changes.

To evaluate the approach, a comparison study against a state-of-the-art syntactic-differencing approach and two line-based differencing tools is conducted as an online within-participant study of about 70 subjects. The results give evidence that srcDiff produces more accurate and understandable deltas.

1.1 Research Questions

The study addresses the following research questions when comparing the different differencing approaches.

- Does srcDiff produce changes that are easier to read and more understandable than the other approaches?
- Is srcDiff preferred over the other approaches?
- Does srcDiff result in a delta that is more accurate than the other approaches?
- Is srcDiff scalable and practical to use in a software development environment?
In addition, this dissertation addresses the following research questions via application of the srcDiff differencing approach.

- How stable are a method’s stereotype?
- What consequences are there to a method’s stereotype changing?
- How commonly do developers replace and rewrite code?
- What are the consequences of replace and rewriting code?
- How often do developers convert code from one type to another?
- What are the consequences of converting code?

1.2 Research Contributions

To the best of the author’s knowledge, no work has presented a syntax-based differencer that completely preserves the underlying source code and thus can show changes to whitespace, comments, preprocessor, and code, works on syntactically incomplete and incorrect code, nor has any work been done for source-code differencing that uses rules based on the source-code syntax as presented here. In addition, to the best of the author’s knowledge, no previous work has done a full user study on viewing source-code changes. An exception is GumTree [Falleri et al. 2014], which performed a manual investigation with the authors and is related to the manual investigation used to develop srcDiff’s rules, and Dotzler and Philippsen [Dotzler, Philippsen 2016] that performed a questionnaire with few participants (eight participants) that are not all computer scientists and only using optimal tree-differencers. The following is a summary of the contributions:

- Development of differencing rules based on knowledge of how developers change code and perceive changes
• Integration of these rules into an efficient syntactic-differencing tool
• First user study on viewing and evaluating differencing approaches with a large number of participants
• Development of a differencing approach that produces more readable, understandable, and accurate deltas as compared to other approaches
• Investigation into the stability of method stereotypes and their consequences via a detailed analysis of changed methods
• Investigation into the prevalence of code conversion and code rewrite/replacement and the consequences

1.3 Organization

The dissertation is organized as follows. Background and related work is presented in CHAPTER 2. The infrastructure supporting the srcDiff approach is presented in CHAPTER 3. CHAPTER 4 introduces the srcDiff approach and describes the algorithm along with the rules we developed. The srcDiff tool is discussed in CHAPTER 5. A preliminary study is described in CHAPTER 6 and the evaluation is presented in CHAPTER 7. In CHAPTER 8 and CHAPTER 9, two applications of srcDiff to software engineering analysis tasks are presented. CHAPTER 8 investigates changes in method stereotypes and CHAPTER 9 investigates code replacement and code conversion. The conclusion of the dissertation is presented in CHAPTER 10.
CHAPTER 2

Background and Related Work

This section presents some basic terminology about differencing in relation to source code.

Differencing is the process of taking any two versions of code and computing the changes between them. For ease of explanation, we will discuss these in terms of successive versions. That is, there is an original version of the source code, and the source code is changed in some manner to produce a modified version.

The individual changes to source code are known as edits. Here, we define an edit as an indivisible operation that transforms the source code. There are two basic types:

- delete – an edit that removes something from the source code.
- insert – an edit that adds something to the source code.

Several more composite edit types also exist which can be described in terms of these basic edit types.

- substitution – an edit where source code in the original is replaced with code in the modified. That is, source code is deleted (a delete edit) in one location, and source-code is inserted (an insert edit) at the same location.
- move – an edit where source code is moved from one location to another. That is, source code in the original version is deleted (delete edit) and inserted (insert edit) at a different location.
In addition, there is a concept of a change or replace. A change or replace is one or more delete edits and one or more insert edits at the same location. This is very similar to the description of a substitution edit, however, a change is not an edit, but a description about a collection of edits. That is, an algorithm may compute substitution edits, while another that does not, reports the same operation as a delete edit and an insert edit (i.e., a change/replace). In this document, we will use “substitution” when we mean the edit version, while we will use replace or change to mean the other.

Depending on how the source code is represented, a single edit of code will take on different forms. That is, when textual line differencing is applied, source code is treated as a sequence of lines and a single edit would be to one line of source code such as one deleted line or one inserted line. When syntactical differencing is performed, the source code is represented as a tree, and the edits are to nodes of the tree such as a deleted node or inserted node. We now talk about textual, syntactical, and other types of differencing and the relevant research on each. We will then discuss topics related to source-code differencing and the other topics used as part of this dissertation. This includes: version-control systems, software merging, and source-code representations.

2.1 Textual Differencing

In textual differencing, the code being differenced is represented as a sequence (e.g., lines, words, or characters). Computing the difference between the original and modified is calculated by finding the minimum number of edits to transform the one sequence into the other (i.e., shortest edit script). There are several different algorithms for computing the shortest edit script, with variations performing better under various
conditions. For example, one of the most popular algorithms is Myers’ algorithm [Myers 1986] which can compute the shortest edit script in $O(ND)$ time, where $N$ is the sum of the lengths of both the original and modified sequences and $D$ is the shortest number of edits. Thus, it is highly efficient (approaches a linear algorithm) the more similar the two sequences. This is ideal for source code, as much of the code remains unchanged between versions. The normal algorithm utilizes $O(N^2)$ space, but applying a Hirschberg linear refinement can turn it into $O(N)$ space [Myers 1986]. The popular GNU diff utilizes Myers’ algorithm with the Hirschberg linear refinement in combination with heuristics [Free Software Foundation 2010]. Figure 2.1 shows two sample outputs of GNU diff on the function `setImage` (simplified to fit) from KOffice (revisions 1026809-1026810). The original version of the function (top-left) and modified version of the function (top-right) are also provided. The middle is a side-by-side view of the changes between the two functions. For GNU diffs side-by-side view, the original and modified function are place next to each other and the changes are indicated via characters in the center. A `<` means the line on the left was deleted (none in example), and `>` means the line on the right was inserted. A `|` means the line on the left changed to the line on the right (i.e., the line on the left was deleted and the line on the right was inserted). On the bottom is a GNU diff’s unified view. In a unified view, the original and modified functions are merged and shown together. Deleted lined are preceded by a `–` and insert edits are preceded by a `+`. 
Additional shortest edit script algorithms include the following. Wagner and Fischer provides a $O(N^2)$ time and space algorithm [Wagner, Fischer 1974] and a subsequent refinement by Hirschberg solved the problem with linear space [Hirschberg 1975]. An approach by Masek and Paterson allows for a $O(N^2 \log \log N / \log N)$ approach for arbitrary alphabets and a $O(N^2 / \log N)$ for finite alphabets [Masek, Paterson 1980]. Let $L$ be the length of the longest common subsequence between two sequences, then Hirschberg provided two additional algorithms taking $O(NL + N \log N)$ and $O(DL \log N)$ respectively [Hirschberg 1977]. Finally, if we let $R$ be the total number of ordered pairs of positions for which the two input sequence match, then, an algorithm by Hunt and Szymanski has a time complexity of $O((R + N) \log N)$ [Hunt, Szymanski 1977]. As mentioned in the previous paragraph, since code is typically modified in small increments ($D$ is small), Myers’ algorithm will have a better run-time performance.

---

Figure 2.1. The output of GNU diff on a method `setImage` (simplified to fit) from KOffice revisions 1026809-1026810. On the top is the original and modified function. The middle is the side-by-side view, and the bottom is the unified view.

Additional shortest edit script algorithms include the following. Wagner and Fischer provides a $O(N^2)$ time and space algorithm [Wagner, Fischer 1974] and a subsequent refinement by Hirschberg solved the problem with linear space [Hirschberg 1975]. An approach by Masek and Paterson allows for a $O(N^2 \log \log N / \log N)$ approach for arbitrary alphabets and a $O(N^2 / \log N)$ for finite alphabets [Masek, Paterson 1980]. Let $L$ be the length of the longest common subsequence between two sequences, then Hirschberg provided two additional algorithms taking $O(NL + N \log N)$ and $O(DL \log N)$ respectively [Hirschberg 1977]. Finally, if we let $R$ be the total number of ordered pairs of positions for which the two input sequence match, then, an algorithm by Hunt and Szymanski has a time complexity of $O((R + N) \log N)$ [Hunt, Szymanski 1977]. As mentioned in the previous paragraph, since code is typically modified in small increments ($D$ is small), Myers’ algorithm will have a better run-time performance.
Ldiff is an extension to GNU diff that adds enhancements to identify fragment changes and moves. Its output is similar to GNU diff [Canfora, Cerulo, Penta 2009]. Both GNU diff and Ldiff can be used in conjunction with a diff viewer such as TkDiff [2011], which will highlight within-line changes. spiff produces results similar to GNU diff, but highlights tokens within a line that changed. Google-diff-match-patch computes changes on a character basis and colors the changed tokens. As all these tools are textual, the differences they report do not always make sense syntactically. The approach proposed in this study is compared to several popular textual difference tools that support within line changes (GitHub’s differencing view [GitHub 2016] and Mergely [Peabody 2016]). The results show that the approach presented in this study outperforms textual differencing approaches along all investigated criteria.

2.2 Syntactic Differencing

Syntactical differencing approaches utilize the Abstract Syntax Tree (AST), a tree representation of code where nodes in the tree contain information about the syntax and structure of the language. The problem of computing the difference between two versions of source code is then finding the difference between the two trees. Typical, approaches compute the shortest edit script (smallest number of edits to nodes that will transform the one tree into the other) between the two trees. When considering arbitrary move edits (code was deleted in one location in the tree and inserted in another location), the problem is NP-Hard [Bille 2005]. Thus, approximation techniques are utilized. General algorithms for the tree-edit distance problem include the work of Chawathe et al. [Chawathe, Rajaraman, Garcia-Molina, Widom 1996], Zhang and Shasha [Zhang, Shasha 1989], and
Pawlik and Augsten [Pawlik, Augsten 2011]. Although, these algorithms are not directly used for syntactical differencing, they have been used as either the basis or as part of syntactic-differencing approaches.

In [Fluri, Wursch, Pinzger, Gall 2007], Fluri et al presents ChangeDistiller, a tree-differencer, which computes an approximate minimal edit script (via computing similarities) that is used to classify and report (output) the tree-edit operations according to their taxonomy [Fluri, Gall 2006]. The algorithm is based off of Chawathe et al.’s algorithm [Chawathe, Rajaraman, Garcia-Molina, Widom 1996]. ChangeDistiller is designed to work on a simplified tree where leaf nodes are statements. GumTree [Falleri et al. 2014] is another tree-differencing approach that approximates a shortest edit script. GumTree compared itself to ChangeDistiller both on the ChangeDistiller tree and a fuller-tree and was shown to produce a shorter edit script in most to nearly all cases. In addition, GumTree was compared to RTED (i.e., Pawlik and Augsten algorithm) [Pawlik, Augsten 2011], a tree-differencer that produces an optimal edit-distance when moves are not considered, and was found to produce a shorter edit script in a majority of the cases. Pawlik and Augsten algorithm [Pawlik, Augsten 2011] is also used internal to GumTree. GumTree supports several outputs including a human readable side-by-side view by overlaying changes onto the source-code. In the evaluation in this paper, we compare the presented approach (i.e., srcDiff) with GumTree, and show that the presented approach is superior on all investigated criteria. In [Dotzler, Philippson 2016], GumTree, ChangeDistiller, and other differences are improved to produce even shorter edit scripts. They also present MTDIFF, which is based on ChangeDistiller, but has better move detection.
Dotzler and Philippsen [Dotzler, Philippsen 2016] was not yet published at the time of our evaluation or original write up. A goal of this dissertation is to show that obtaining a shortest edit script fails as a good metric for computing the difference between two versions of source code. So, as their goal is to create an even shorter edit script, this would only exasperate the problems identified in this dissertation.

diff/TS [Hashimoto, Mori 2008] is a syntactic-differencing approach that like GumTree aims to approximate an optimal tree-edit distance. It utilizes Zhang and Shasha’s algorithm [Zhang, Shasha 1989]. It is also one of the few syntactic-differencing approaches that provides a human-readable view. It does this by overlying edit operations onto the source-code text. As GumTree is more recent, and it compared itself to other well-known source-code differencers, GumTree is chosen. As diff/TS also approximates an optimal tree edit distance, the results of our study generalize.

JSync [Nguyen et al. 2012] is a clone management tool for evolving software that incorporates an approximate optimal tree edit algorithm of their own design and LTDIFF [Hunt, Tichy 2002] simulates tree differencing for use with their software merger. As part of both approaches (as well as a very early version of srcDiff [Maletic, Collard 2004]), they first use textual differencing and map/align the changes to text with an AST to come up with a tree difference. Both approaches could be used to report a human readable difference. However, this was not their original intent and since, the approaches are doing some form of approximation of optimal tree-differencing the results of this dissertation still apply.
Smart Differencer [Semantic-Designs 1995-2012], based on Baxter’s Design Maintenance System (DMS) [Baxter, Pidgeon, Mehlich 2004], is a heavy-weight compiler-based approach to source-code parsing and syntactic differencing of source code which describes the changes in terms of abstract editing operations and gives the code fragments affected. It reports changes to preprocessor and comments, but not to whitespace. Smart Differencer is proprietary software.

The most recent version of srcDiff appeared in a Master’s thesis [Decker 2012] and is similar only in name to [Maletic, Collard 2004]. Additionally, the rules presented here are a significant extension to the thesis.

2.3 Semantic Differencing

Semantic differencing is a form of source-code differencing that considers the semantics of source code and reports changes in terms of what changed in behavior. That is, source code may be modified textually and syntactically, however, the original code and modified code may be semantically equivalent. In such a case, a differencer supporting full semantic detection would report no change. In practice, however, semantic differencing is an undecidable problem [Berzins 1986; Jackson, Ladd 1994], so current semantic differencers are limited to some extent. The following are semantic-differencing approaches.

Dex is a syntax and semantic source code difference tool that uses graph differencing on Abstract Semantic Graphs (ASG) created utilizing patches between two systems. The output is a properties file describing the changes in a patch [Raghavan, Rohana, Podgurski, Augustine 2004]. JDiff is a diff tool for Java that understands object-
oriented features and outputs a pair of nodes sets and their status (modified/unchanged) [Apiwattanapong, Orso, Harold 2004; Apiwattanapong, Orso, Harold 2007]. Due to what these tools provide as output, they were not directly comparable with srcDiff, and so were not included as part of the evaluation.

Semantic Diff performs a difference using the change of dependence relations between inputs and outputs of a procedure and results in a textual description [Jackson, Ladd 1994]. Semantic Diff presents changes in a separate manner than srcDiff, and can be used in conjunction with the approach in this dissertation.

2.4 Track-changes Approaches

Omori and Maruyama [Omori, Maruyama 2008] propose an IDE that records edit operations as they occur. Similarly, SpyWare is a change-aware development toolset that captures changes as they occur and can provides many views of source-code change, including a session inspector that allows changes in a session to be replayed [Robbes, Lanza 2008]. Dias [Dias 2015] gives another example of an approach that listens and records developer actions taken in an IDE. The main difference is these approaches record changes as they are made, while srcDiff, is able to recover changes without having a specially designed development environment.

2.5 File Comparison Software

A multitude of file comparison software exists. A list of these can be found at [Wikipedia® 2017]. Examples include TkDiff [2011], Meld [Willadsen 2011-2012], diff
and diff 3 [Free Software Foundation 2010; Miller, Myers 1985], Kompare [Keel, Snyder 2013], and Eclipse (compare) [Foundation 2017].

2.6 Other Differencing Approaches

There is a considerable number of other differencing approaches that are less related to the work presented here. Since, srcDiff works on an XML (eXtensible Markup Language) [W3C 2008] representation of source-code, the most relevant are XML-differencing techniques [DeltaXML 2000-2016; IBM 1998; INRIA ; Wang, DeWitt, Cai 2003]. XML-differencing techniques are designed for general XML documents, that is, unlike srcDiff, they have no knowledge of the underlying source code or AST and its significance for differencing.

LSdiff infers systematic structural differences as logic rules [Loh, Kim 2010]. Since, this approach reports differences in a separate manner than what is presented here, it can be used in conjunction with a tool like srcDiff.

Additionally, there are separate bodies of work that difference non-source code such as UML [Zhenchang, Eleni 2005] and Verilog hardware descriptions [Duley, Spandikow, Kim 2012].

2.7 Version Control Systems

Tichy defines Software Configuration Management (SCM) as the discipline of managing the evolution of large and complex software systems [Tichy 1988]. An integral part of this is version control systems. In their book, Pro Git, Chacon and Straub define Version Control Systems (VCS) or version control as systems that records changes to a file
or set of files over time so that you can recall specific versions later [Chacon, Straub 2014]. Examples of Version Control Systems are: Git [Inc.], Subversion (SVN) [Foundation 2011], Mercurial, Microsoft’s Team Foundation Server [Microsoft 2017], Perforce [Perforce 2017], and Concurrent Versions System (CVS) [mdb 2015]. They are integral to the software development process as they enable and support the collaboration of software developers on software projects. As part of VCS, differencing tools are usually included. For example, SVN incorporates a textual line diff, and Git includes several textual diffs: line, word, and a separate algorithm named “patient diff” [Cohen 2010]. In addition, VCS can incorporate differencers as part of their storage mechanism. That is, by storing deltas (changes) relative to a base version, a VCS can save a substantial amount of disk space [Hunt, Vo, Tichy 1998].

2.8 Software Merging

Highly related to the field of software differencing is software merging. For a more complete discussion on software merging than discussed here, see Mens [Mens 2002]. As software development is a significantly collaborate effort, developers work on separate bodies of code in parallel. Periodically, these parallel lines of development need to be merged into one. Software merging is the process of integrating together parallel lines of development of the same software. As the parallel versions are typically kept in the same Version Control Systems (VCS), they usually incorporate mechanisms for merging. In addition, merging software requires knowing what is the same and what is different between the two parallel versions. As such, software differencing is an integral part of
software merging. To simplify discussion, we will discuss software merging in relation to merging two parallel lines of development.

There are two types of merging techniques: two-way merging and three-way merging. In two-way merging, two files are merged by comparing the differences between the two parallel versions. In three-way merging, a third file, a common ancestor, is used to clarify edits (i.e., what was truly inserted/deleted). Due to the additional power of three-way merges, they are used almost exclusively [Mens 2002].

Analogous to differencing techniques, there are also textual, syntactic, and semantic merging techniques. As with differencing, textual merging treats the software as text (e.g., lines), syntactic merging incorporates elements of the syntax of the language, and semantic merging uses behavior elements as part of the merge. With the progression from simple textual merging to syntax merging through to semantic merging techniques, more conflicting edits between the parallel versions can be detected and less important conflicts such as to comments or whitespace can be ignored [Mens 2002].

GNU diff [Free Software Foundation 2010] provides support for textual two-way merging and GNU diff3 [Free Software Foundation 2010] provides support for three-way merging [Mens 2002]. In addition, VCS such as Git [Inc.], Subversion (SVN) [Foundation 2011], and Concurrent Versions System (CVS) [mdb 2015] provide or incorporate textual merge tools. Despite the simplicity of textual-merge techniques, according to a case study, they are able to automatically merge roughly 90% of files without conflict [Perry, Siy, Votta 2001]. The reason why textual merges fail in the 10% of cases is due to a lack of syntactic and semantic information [Mens 2002]. In addition, the study notes that in 1% of
all cases, the tools made an incorrect automatic merge, usually resulting in compiler errors [Perry, Siy, Votta 2001], which can be solved by adding syntactic information.


Semantic merging adds semantic information to arrive at a better merge. Often, graph-based approaches are used, as an AST does not contain enough context information for a semantic merge [Mens 1999]. An exception to this is Westfechtel’s approach [Westfechtel 1991] which augments the AST with some context information. Hunt and Tichy’s approach (ELAM: Extensible Language-Aware Merging) [Hunt, Tichy 2002], although not quite a semantic merger, is a tree-based three-way merger that utilizes some semantic information. Mens [Mens 1999] is an example of a graph-based approach.

To avoid unexpected behavior in the merged result, more complex semantic-merging approaches add additional formalisms [Mens 1999]. Berzins uses denotational semantics [Berzins 1994] for merging, Horwitz et al. use both program slicing and program dependency graphs [Horwitz, Prins, Reps 1989], Yang et al. use what they call limited slicing and program-representation graphs [Yang, Horwitz, Reps 1992], and Binkley et al. generalize Horwitz et al.’s approach to languages with procedure calls with interprocedural slicing [Binkley, Horwitz, Reps 1995].
2.9 Source-code Representations

The work of this dissertation builds off of and adds to the srcML Infrastructure [Collard, Decker, Maletic 2011; Collard, Decker, Maletic 2013]. The most closely related work to srcML is JavaML [Badros 2000]. Although JavaML provides an alternative representation of Java source-code, it does not preserve the source code, a significant advantage to using srcML. Examples of other XML representation are GXL [Holt, Winter, Schürr 2000], CppML [Mammas, Kontogiannis 2000], Aterms [van den Brand, Sellink, Verhoef 1998], GCC-XML [Kitware 2002-2012] (which has been succeeded by CastXML [Kitware 2013-2015]), and Harmonia [Boshernitsan, Graham 2000]. In these, the AST is encoded in XML and not the actual source code.

2.10 Improving Upon the Literature

A deep review of the literature identified the following. Textual-differencing approaches are by far the most commonly used and available, but they have problems as they lack information about the syntax of the language. Semantic differencing is attractive as it can be used to ignore semantically equivalent changes, however, the approaches are computationally expensive, and the general problem is undecidable [Berzins 1986; Jackson, Ladd 1994]. Track-changes approaches are capable of capturing exactly what edit events occur, but their use is restrictive and they cannot be applied in a general case. Syntactic-differencing approaches, therefore, have the greatest potential.

As syntactic-differencing approaches include the program syntax and structure, the resulting edits can conform to that syntax and structure. However, current syntax-
differencing approaches have several shortcomings, which inhibit them from seeing widespread use.

- Rely on optimal tree-edit distance as metric
- Under-utilization of program syntax
- Discard comments, whitespace, and/or preprocessor statements
- Lack a human readable view of source-code

First, optimal tree-edit distance is used to compute the difference. That is, approaches aim at shortening the tree-edit script. This dissertation will show in CHAPTER 4 and in the evaluation (CHAPTER 7) that this metric is flawed. A shorter edit script does not lead to a better or more accurate difference.

Secondly, the syntax of the language is not used to the full extent. Nodes in the tree are labeled with the syntax (i.e., a node is an if-statement) and an algorithm to do tree differencing is applied. Rarely is the encoded syntax or semantics of the syntax used beyond the matching of labels by the algorithm.

Third, many approaches discard whitespace, comments, etc. in the tree and so are unable to produce the difference of these elements. Tied to this is the fourth shortcoming. Since, this whitespace, comments, etc. are not recorded, they cannot go from the AST to the original source-code to produce a human-readable difference. Those that do resort to other means such as recording character positions as part of the nodes when the AST is formed so that they can overlay changes onto the source code, which has limitations.

The approach presented in this dissertation, srcDiff, addresses these shortcomings by:
• Purposefully deviating from an optimal tree-edit distance to result in a better difference.

• Utilize a set of rules based on the program syntax and domain knowledge on how developer’s change and perceive code change.

• Preserving whitespace, comments, and preprocessor statements.

• Providing human readable views of source-code
CHAPTER 3

srcML Infrastructure

The srcML Infrastructure is used extensively as a part of this dissertation. As such, the following is an in-depth discussion of the infrastructure. The srcML Infrastructure is a format and a set of tools and technologies to support the exploration, analysis, and manipulation of large software projects [Collard, Decker, Maletic 2011; Collard, Decker, Maletic 2013; Kagdi 2003]. That is, it supports the maintenance and evolution of real software. Figure 3.1 is a diagram (from srcML.org) of the srcML Infrastructure. The srcML Infrastructure is comprised of the following parts: srcML format, srcML, XML, Models, Tools, and Support. The srcML format is an XML (eXtensible Markup Language) representation of source-code that forms the foundation of the infrastructure. The tool srcml converts source code and software systems to and from the srcML format. XML technologies and additional models are used to develop tools supporting software evolution and maintenance (i.e. the exploration, analysis, and manipulation of source-code). Tying every together is the Support of a multi-university development team. For the remainder of this chapter, the parts of the srcML Infrastructure will be discussed in detail.
Figure 3.1. Diagram of the srcML Infrastructure. The foundation is the srcML Format. On top of this is the srcML tool, which converts source code to and from the srcML Format. From this core functionality, XML technologies and additional models can be leveraged to develop tools to support exploration, analysis, and manipulation of large software systems. The infrastructure is supported (right) via a multi-university development team.

3.1 srcML Format

The srcML format is an XML representation of source code. In the srcML format, all the original source code text (i.e., code, preprocessor statements, and whitespace) is marked up (i.e, wrapped) with XML tags (e.g., start tag: <expr>, end tag: </expr>) that represent information about the syntax and structure of the source code. This information is known as an Abstract Syntax Tree (AST). Figure 3.2 provides an example function (top) and its markup in the srcML format (bottom). For comparison between the source code and srcML format representations, the original text is highlighted in bold in the srcML format representation. At the top, is the root tag unit, which stands for compilation unit (i.e., file) and is common to all srcML documents. In addition to being
the root document, it also encodes metadata (as XML attributes) about the file and how it was created such as what version of srcML produced the file (revision="0.9.5") and the input file’s name (filename="original.cpp"). Internal to the unit tag information about the program syntax is marked up. For example, the function is marked with a function tag and the expression statement in the body of the function is wrapped with an expr_stmt tag.

A significant advantage to using the srcML format is that all the original source code is preserved. This is important because any resulting transformation of the code that is intended to be returned to a developer will be rejected unless the programmer’s view (i.e., whitespace, comments, etc.) of the source code is maintained [Cordy 2003; Van De Vanter 2002]. Since the srcML format is lossless, removing all the XML tags, along with some unescaping, will reproduce the original source-code.

<table>
<thead>
<tr>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Settings::setImage( KisImageSP image ) {</td>
</tr>
<tr>
<td>m_widget-&gt;m_filter-&gt;setImage( image );</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>srcML</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;unit xmlns=&quot;http://www.srcML.org/srcML/src&quot; xmlns:cpp=&quot;http://www.srcML.org/srcML/cpp&quot; revision=&quot;0.9.5&quot; language=&quot;C++&quot; filename=&quot;original.cpp&quot;&gt;</td>
</tr>
<tr>
<td>&lt;parameter_list&gt;</td>
</tr>
<tr>
<td>&lt;/parameter_list&gt;</td>
</tr>
<tr>
<td>&lt;block&gt;</td>
</tr>
<tr>
<td>&lt;expr_stmt&gt;&lt;expr&gt;&lt;call&gt;&lt;name&gt;&lt;name&gt;m_widget&lt;/name&gt;&lt;operator&gt;-gt;&lt;/operator&gt;&lt;name&gt;m_filter&lt;/name&gt;&lt;operator&gt;-gt;&lt;/operator&gt;&lt;name&gt;setImage&lt;/name&gt;&lt;argument_list&gt;{ &lt;argument&gt;&lt;name&gt;image&lt;/name&gt;&lt;/argument&gt;&lt;/call&gt;&lt;/expr&gt;</td>
</tr>
<tr>
<td>&lt;/block&gt;&lt;/unit&gt;</td>
</tr>
</tbody>
</table>

Figure 3.2. Example source code function and its markup in srcML. Syntax information such as a section of code being a function or an expression statement is encoded by wrapping the appropriate source code with XML tags. The process is lossless. All the original text, including whitespace and comments is preserved.

Figure 3.2 shows a single file in the srcML format. The srcML format additionally supports the storage of entire software projects (i.e., a collection of files) in a single
document. This is known as a srcML archive. Figure 3.3 shows an example of a srcML archive. The example consists of two source code files: `sum.hpp` and `sum.cpp` in the srcML format. A separate `unit` tag is added around both file `unit` tags to comprise a srcML archive. Each file `unit` tag contains the original file’s path and name (e.g., `filename="sum/sum.hpp"`).

```
<unit xmlns="http://www.srcML.org/srcML/src" revision="0.9.5">
  <unit xmlns:cpp="http://www.srcML.org/srcML/cpp" revision="0.9.5" language="C++" \
  filename="sum/sum.hpp">  </unit>
<unit xmlns:cpp="http://www.srcML.org/srcML/cpp" revision="0.9.5" language="C++" \
  filename="sum/sum.cpp">  </unit>
</unit>
```

**Figure 3.3.** Example srcML archive. A separate `unit` tag is added as a root tag around the file `unit` tags

### 3.2 srcML

The tool `srcml` handles the conversion of source code text into the srcML format and the conversion of code in the srcML format back into source-code text.

### 3.3 To srcML Format

`srcml` uses ANTLR (ANother Tool for Language Recognition) [ANTLR 2014] to parse source code and produce the srcML format. Currently, `srcml` supports the conversion of C, C++, C#, and Java into the srcML format. In addition, Objective-C is unofficially supported and a mechanism for plugging-in any arbitrary language grammar is under development. The conversion process is remarkably fast. On an iMac running El Capitan with 2.7 GHz Intel Core i5 processor and 8 GB DDR3 RAM, `srcml` can achieve
speeds of ~92KLOC\(^1\) per sec. This means, the entire Calligra [Calligra] project (an Office suite) consisting of about 1,100KLOC can be converted into a srcML archive in less than 12.5 seconds. The remainder of this section gives examples on how to use srcml to convert source-code into the srcML format.

The following will take a source-code file main.cpp convert it into the srcML format and print it to the terminal. The tool uses language extensions to determine what language is contained in the file and how to parse the file. For example, main.cpp has the extension cpp, which corresponds to a C++ file.

```
srcml main.cpp
```

To save the output to a file the \(-o\) option is used. Here the source code file main.cpp is converted to the srcML format and written to the file main.cpp.xml.

```
srcml main.cpp -o main.cpp.xml
```

A srcML archive can be produced by giving a collection of files, a folder, or a compressed/archive file. This first example takes the two files sum.hpp and sum.cpp, converts each separately to the srcML format and writes them out into sum.xml as a srcML archive.

\(^{1}\) KLOC – thousand lines of code.
Here, a folder (and all its subfolders) containing source-code is traversed. Each source code file is converted into the srcML format and written out to the file `project.xml` as a srcML archive. Language extensions are used to determine what is a source-code file and how to parse each file. The folder may contain multiple languages.

```
srcml project/ -o project.xml
```

When a compressed and or archive file is passed, `srcml` uses `libarchive` [Kientzle 2003] to un-archive and uncompress the file, walk through all the files contained in the archive and convert each source-code file to the srcML format. In this example, the `tar` archive and `xz` compressed project is processed and all source code files written to `project.xml` as a srcML archive.

```
srcml project.tar.xz -o project.xml
```

### 3.4 From srcML Format

`srcml` uses the Simple API for XML (SAX) [SAX 2001] parser from `libxml2` [Veillard 2004b] to convert code in the srcML format back into the original source code, generating files and folders as necessary. The process is extremely efficient. On an iMac running El Capitan with 2.7 GHz Intel Core i5 processor and 8 GB DDR3 RAM, `srcml`
can achieve a conversion rate of over ~580KLOC per second when converting from the srcML format back to source-code. This means, the entire Calligra project in srcML, can be converted back to source including generating all the files and folders in under 2 seconds. The remainder of this section gives examples on how to use srcml to convert code in the srcML format back into the original source code.

The following takes the srcML format (non-archive) file main.cpp.xml strips out the XML tags and prints the source-code to the terminal.

```
srcml main.cpp.xml
```

The following also strips out the XML tags from the srcML format (non-archive) file main.cpp.xml, but uses the --o option to write the contents to the file main.cpp.

```
srcml main.cpp.xml --o main.cpp
```

The following used the --to-dir option to take a srcML archive and convert each file to the original source-code and write it out into the folder project. The original folder structure is reconstructed in the project folder via the filename attribute on the individual unit tags.

```
srcml --to-dir project/ project.xml
```
3.4.1 libsrcml

In addition to the srcml tool, the srcML Infrastructure provides the library libsrcml. libsrcml provides the core functionality such as recognition of programming language and conversion to/from the srcML format. The srcml tool is then a front-end (i.e., command-line interface) to the library libsrcml. The current implementation of srcDiff (CHAPTER 5) uses libsrcml directly to allow for file-by-file differencing and to avoid the overhead of running an additional application. The srcml tool and the commands to convert to the srcML format functioned as a template for the development of the srcdiff tool. Although, the srcdiff tool replaces the to-srcML format functionality with to-srcDiff format, the from-srcML format functionality of the srcml is still utilized as part of the srcDiff tool chain.

3.5 XML & Models

As the srcML format is an instance of XML [W3C 2008], any XML technology can be applied directly to srcML. For instance, several methods are typically used for the parsing and processing of XML documents such as: Document Object Model (DOM) [W3C 1997], Text Reader [Veillard 2004a], and Simple API for XML (SAX) [SAX 2001]. Searching and querying of XML document is supported, as an example, through XPath [W3C 1999a] and XQuery [W3C 2000], and the transformation of srcML is support, as an example, by XSLT (eXtensible Stylesheet Language Transformations) [W3C 1999b]. The current implementation of srcDiff uses the Text Reader approach from libxml2 to read, and parse the srcML format so that the approach can be applied. In addition, SAX parsing
is used for the parsing of documents in the srcDiff format (such as for generating human-readable output).

The srcML format is extendable. There is no current DTD (document type definition) for the srcML format. That is, there is no specification that states what is or is not valid srcML format. As such, additional models and information can be embedded with or on top of the general srcML format. The srcDiff format is an example. It extends the srcML format with additional tags and attributes. This will be discussed in detail in CHAPTER 4.

3.6 Tools

The srcML Infrastructure is commonly used both in research, academia, and in industry. As such, a number of tools have been produced on top of and integrated into the infrastructure. The research undertaken for this dissertation utilized tools that are a part of the srcML infrastructure and produced other tools that are now incorporated into the infrastructure.

3.7 Support

The development of the srcML Infrastructure is supported via multiple developers at multiple universities. In addition, tutorials and other documentation are freely available at srcML.org. The multi-university developers also provide quick feedback to questions and other information pertaining to using and building on the srcML Infrastructure. This support is invaluable for the research and tool development of this dissertation.
CHAPTER 4

Syntactic Differencing via srcDiff

We developed an efficient syntactic differencing approach based on the srcML [Collard, Decker, Maletic 2013] infrastructure (www.srcML.org). This includes a syntactic-difference output format [Maletic, Collard 2004] based on srcML and differencing tool called srcDiff. The work of this chapter and CHAPTER 7 is accepted with revision to the Journal of Software: Evolution and Process [Decker, Collard, Volkert, Maletic 2017].

4.1 srcDiff Format

The srcDiff format [Maletic, Collard 2004] extends the srcML format with the addition of four XML tags (diff:common, diff:delete, diff:insert, and diff:ws) to contain original and modified source code (i.e., any two versions) marked as the delta or the set of changes to the original source code (base version). Figure 4.1 gives an example of the format, as produced by the srcDiff tool, showing both the original and modified source code (simplified) of the function setImage from KOffice revisions 1026809-1026810. The changes include the statement in the function setImage being wrapped with an if-statement and the data member m_widget being renamed to m_options. The srcDiff subsequently has a diff:insert tag around the if-statement and a diff:common tag around the contents. The text of the renamed identifier (marked in srcML with a name tag) has a diff:delete tag around the old text and a
diff:insert around the new text. In addition, the tags have an attribute type with the value replace to signify that the code was replaced (e.g., rename). If a modification is to an attribute of a srcML element, these values are versioned with |, as in the attribute filename on the unit tag. Deleted/inserted whitespace is marked with diff:ws for easy processing/analysis. The srcDiff format completely preserves the original and modified srcML and therefore completely preserves both the original and modified source code (e.g., code, whitespace, comments, preprocessor). That is, the srcDiff format is a multi-version single document format that allows both the original and modified srcML/source-code to be extracted. As such, srcDiff supports both source-code change analysis, as well as, an efficient means of producing human-readable deltas. Examples of the human-readable format for the changes in Figure 4.1 are in Figure 4.2. The top is known as a side-by-side view, where the original code is presented on the left and modified code on the right. Changes are marked up individually on the original and modified code. For srcDiff, the original method is on the left with deleted code marked with a red background and the modified method is on the right with inserted code marked with a green background. The bottom is a unified view. In a unified view, the original and modified code are merged together and the changes marked on the single merged code. For srcDiff, the original and modified method are merged into one with deleted code marked with a red background and inserted code marked with a green background. As a note, since other syntactic-differencing approaches do not maintain/preserve whitespace, comments and preprocessor statements, they cannot easily (if at all) produce a unified view.
Other syntactic-differencing approaches achieve a side-by-side view by overlaying change information on the text files [Falleri et al. 2014; Hashimoto, Mori 2008].

Typically, syntactic-differencing methods support additional edits (e.g., update node value, move, etc.). Because srcDiff marks up text directly (e.g., renamed identifier in Figure 4.1), it does not need a separate edit for an update. In srcDiff, moves are marked as a delete (moved from) and insert (moved to) tags with an attribute move and a unique identifier.

The srcDiff tool is designed to efficiently produce the srcDiff format from any two versions of a source-code document, i.e., two files, directories, or repository versions (Subversion or Git). In contrast to other syntactic differencers, the code does not need to be syntactically complete, and changes to whitespace and comments are marked up. srcDiff also attempts to produce results on syntactically incorrect code.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Settings::setImage( KisImageSP image ) {</td>
<td>void Settings::setImage( KisImageSP image ) {</td>
</tr>
<tr>
<td>m_widget-&gt;m_filter-&gt;setImage( image );</td>
<td>if (m_options) {</td>
</tr>
<tr>
<td></td>
<td>m_options-&gt;m_filter-&gt;setImage( image );</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Figure 4.1. Example source-code change. The top-left contains the original source code and the top-right contains the modified source code. The original and modified code contained within the srcDiff format is given. Deletions are highlighted in red and with a line-through mark. Insertions are highlighted with green. All the original source code is placed in bold.
The tool is very scalable as it handles 1,000 commits/versions (all changed files) in under 5 minutes (running on a standard desktop computer).

<table>
<thead>
<tr>
<th>Side-by-side View</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void Settings::setImage( KisImageSP image ) {</code></td>
<td><code>void Settings::setImage( KisImageSP image ) {</code></td>
</tr>
<tr>
<td><code>  m_widget-&gt;m_filter-&gt;setImage( image );</code></td>
<td><code>  if (m_options) {</code></td>
</tr>
<tr>
<td></td>
<td><code>  m_options-&gt;m_filter-&gt;setImage( image );</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unified View</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void Settings::setImage( KisImageSP image ) {</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 4.2. An example of the changed methods from Figure 4.1 in two different human readable formats produced by srcDiff. The top is a side-by-side view of the change, where the original method is on the left with deleted code marked with a red background and the modified method is on the right with inserted code marked with a green background. The bottom is a unified view where both methods are merged into one with deleted code marked with a red background and inserted code marked with a green background.

4.2 srcDiff Algorithm

At a high level, srcDiff does a preorder traversal on both the original and modified Abstract Syntax Trees (AST), in srcML, to iteratively refine differences until the desired granularity is reached. A preorder traversal is required so that the resulting difference, in srcDiff format, completely preserves the original and modified code. Figure 4.3 presents the differencing algorithm for srcDiff. The algorithm applies a sequence-differencing algorithm (i.e., Myers’ [Myers 1986]) to the original and modified children (including subtrees) of a node (see line 2). The result of the sequence differencing is a list of edits
where each edit is one or more consecutive children of the same edit type. The edit types are:

- **common** - children identical in both versions,
- **delete** - children removed from the original AST,
- **insert** - children inserted into the modified AST, and
- **change** - children from the original AST replaced with children from the modified AST\(^2\).

Common and unique children are output directly (lines 5-9), however, changed children need to be further analyzed to determine which actions to take (lines 10-26). There are three possible situations:

- **Match** - the child in the modified is a new version of the original;
- **Nested** - one child and its subtree in one version subsumes children in the other version;
- **Deleted** and/or **Inserted** - a child is unique to one of the versions.

Figure 4.4 illustrates how srcDiff uses sequential differencing. Part a) gives the original and modified versions of a declaration statement, as well as, their tree representations. For the tree representation, only the declaration statement subtree is shown. That is, the root is the `decl_stmt` and the declaration statement’s children are:

\[
\text{declaration statement subtree}
\]

\(^2\) A change can also be described as a list of deleted children and a list of inserted children that occur at the same relative location.
type (type) and its subtree, identifier name (name) and its subtree, and initialization (init) and its subtree. The original declaration statement (source code) was modified in two ways: the addition of the extern keyword and the replace of the numeric literal 42 with the constant ANSWER. As such, the tree representation is modified in two ways: a new child of decl_stmt (specifier and its subtree) is added and the literal (descendent of init) is replaced with the constant ANSWER (i.e., name). srcDiff takes the children of a common node (decl_stmt), treats each as a sequence, as shown in Figure 4.4 part b), and runs a sequential-diff algorithm (i.e., Myers’ [Myers 1986]). The result of this for Figure 4.4 is part c). The specifier and its subtree is reported as an insert edit. In addition, the init (and its subtree) is reported as a delete edit in the original tree and an insert edit in the modified tree. As they occur in the same relative position in the sequence, the init insert and delete edits get aggregated into a change edit. The remaining children are then denoted as common. The insert of specifier and common children require no further action by the srcDiff algorithm. However, the changed children (i.e., init), are analyzed further to determine if they were Inserted/Deleted (i.e., original init was deleted and the modified init inserted), Nested (one init belongs inside the other), or a Match (i.e., deleted init is a previous version of the inserted).

The determination of which actions to apply are as follows. First, the changed children are checked to determine if any of the children can be matched (i.e., using the function determineMatchings). If so, then finer-grained differencing is performed via a recursive call to the srcDiff algorithm on their children. The determineMatchings process uses a dynamic-programming approach that minimizes
the number of unmatched children while utilizing a similarity metric as a tie breaker. What can be matched is determined via a set of differencing rules (discussed in detail in the following section). The result of determineMatchings is a sequence of items that is either consecutive original and modified children with no corresponding matches (i.e., unmatched) or a pair of matched children.

Next, unmatched children are checked (via determineNestings at line 16) to determine if any of the children in the original or modified AST can be nested within the other. The process works by iteratively examining each original child to determine if the modified children are nested within. This process is duplicated for the modified children to see if any of the original children are nested within. Similar to determineMatchings, a set of differencing rules is used to define the nestings. If only the original or modified has a nested child, that nesting is chosen. If both situations produce a possible nesting and the nestings have no common children, the nesting with the first occurring children is selected. When the original and/or modified nestings share children, the nesting that covers the largest number of children is selected. This process continues until no new nesting is found. The result of determineNestings is a list that partitions all the children into groups of consecutive un-nested children or children pertaining to a nesting. For each nesting, the deleted/inserted root is output and the srcDiff algorithm recursively called on the children.

In the case that a child can be both matched and nested, determineMatchings uses the differencing rules to determine if nesting produces a better result. If so, the match is rejected and the children will be left as unmatched for determineNestings. In
addition, `determineNestings` is greedy, selecting the first possible child that can be nested. In practice, we found this to work well, except in the case when there is a single child deleted or inserted and nothing can be nested into it, but it can be nested. In this case, `srcDiff` does an exhaustive search to determine the best matching.

Lastly, is the situation of a delete or insert; this can be output directly (see line 22). As a note, `srcDiff` provides limited support for moves (see line 3). Children of the original that are identified as deleted or changed by the sequence differencer are scanned to see if there is an exact match among the modified children identified as inserted or changed. If found, the children are marked as moves and ignored by `determineMatchings` and `determineNestings`. Further support for moves can be added by analyzing the resulting `srcDiff`, however we leave this for future work. Next, we turn to discussing the differencing rules.
1 srcDiff(original, modified)
2 ListOfEdits = computeShortestEditScript(original, modified)
3 markMoves(ListOfEdits, original, modified)
4
5 FOR EACH edit IN ListOfEdits DO
6   IF common THEN
7     OUTPUT children as common, also merging whitespace
8   ELSEIF deleted OR inserted THEN
9     OUTPUT deleted or inserted children marked appropriately
10  ELSE //a change
11     matches = determineMatchings(edit.original, edit.modified)
12     FOR EACH i IN matches DO
13       IF i is matched THEN
14         OUTPUT i.root
15         srcDiff(i.originalChildren, i.modifiedChildren)
16       ELSE //Apply nesting rules to a sequence of unmatched children
17         nestings = determineNestings(i.originalChildren,
18                                  i.modifiedChildren)
19         FOR EACH j IN nestings DO
20           IF j is nested THEN
21             OUTPUT deleted or inserted root marked appropriately
22             srcDiff(j.originalChildren, j.modifiedChildren)
23           ELSE
24             OUTPUT deleted and inserted children marked appropriately
25           END IF
26         END FOR
27       END IF
28     END FOR
29   END IF
30 END FOR
31 END

Figure 4.3. The srcDiff algorithm. A preorder traversal is performed on both the original and modified trees. The input is the original and modified children of a node. A sequence-differencing method is used to categorize the children as common, unique to one version, or changed. Common, inserted, and deleted children are output directly. Changed children are further analyzed in determineMatchings and determineNestings for the following situations: a child in the original is a previous version of a child in the modified, a child and its subtree in one version subsumes children in the other version, and a child is unique to one version. The functions determineMatchings and determineNestings use a set of rules to identify these situations.
Figure 4.4. Example of the srcDiff process. a) The original and modified version of a declaration statement with their tree representations below each. b) With \texttt{decl\_stmt} as common root, run a sequence diff on their child sequences (\textcolor{blue}{blue}). Compares are between entire child subtrees. c) Output of sequential diff. \textcolor{red}{red} is delete edit, \textcolor{green}{green} is insert edit, remaining are common. \texttt{<specifier>} subtree is an insert and \texttt{<init>’s} are a change. Common and insert/delete require no further action. The change needs to be analyzed to see if \texttt{Match}, \texttt{Nested}, or \texttt{Inserted/Deleted}. 
4.3 srcDiff Rules

Pure syntactic tree differencing often produces results that are difficult to understand. Take the following example code from hadoop (revision: eee0d4563c626a7c0f3a6d05ee713aaf69add78):

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static boolean isStoragePolicyXAttr(XAttr xattr) {</td>
<td>public static String getStoragePolicyXAttrPrefixedName() {</td>
</tr>
<tr>
<td>return xattr != null &amp;&amp; xattr.getNameSpace() == XAttrNS &amp;&amp;</td>
<td>return XAttrHelper.getPrefixedName(XAttrNS, \</td>
</tr>
<tr>
<td>xattr.getName().equals(STORAGE_POLICY_XATTR_NAME); } }</td>
<td>STORAGE_POLICY_XATTR_NAME);</td>
</tr>
</tbody>
</table>

Textually, there is only a small amount of similarity between the two versions. Clearly, the method in the original version is completely replaced with a new method in the modified version that is, a delete and insert. Using a traditional syntactic differencing approach, it is more optimal (i.e., shorter edit script) to match any two given constructs (e.g., methods, if-statements, declaration-statements, etc.) because of common AST nodes that are shared. However, this approach fails miserably in this type of example and results in something very difficult to understand. The following is a side-by-side difference produced by GumTree [Falleri et al. 2014] a current state-of-the-art syntactic differencing tool. Deletes are marked in red, inserts in green, moves in blue. Updates are given in gold with common text in grey and deleted/inserted text in bold.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static boolean isStoragePolicyXAttr(XAttr xattr) {</td>
<td>public static String getStoragePolicyXAttrPrefixedName() {</td>
</tr>
<tr>
<td>return xattr != null &amp;&amp; xattr.getNameSpace() == XAttrNS &amp;&amp;</td>
<td>return XAttrHelper.getPrefixedName(XAttrNS, \</td>
</tr>
<tr>
<td>xattr.getName().equals(STORAGE_POLICY_XATTR_NAME); } }</td>
<td>STORAGE_POLICY_XATTR_NAME);</td>
</tr>
</tbody>
</table>

A large number of meaningless matches between syntactic elements (e.g., XAttr and String) are made to achieve more optimal results (i.e., shorter edit script). While
entire method replacement (as in this example) may not be very common, replacement of
statements (or parts of statements) is a very common type of change. Anytime there is a
coincidental similarity, pure syntactic differencing approaches will generate an
inappropriate result.

Hence, we take a different approach here and use a set of customized rules instead
of a single heuristic such as edit-script distance. We defined a set of rules that determine
when it is acceptable to consider whether the change is a modification or a complete
replacement of an existing statement. We derived the rules by manual and statistical
analysis of over 1,000 revisions/version of changes to several open-source projects along
with our combined development experience. Constant values within the rules are derived
through statistical analysis and fine-tuning (via trial and error). These values can be further
adjusted based on preference or additional empirical studies. In addition to the empirical
analysis, we tested srcDiff on a suite of over 500 examples (simple to very complex) that
includes changes from various open-source systems.

The differencing rules employed by srcDiff are organized into three categories:

- *Match* rules - identifies when two syntactic elements of the same type (e.g.,
two if statements) can be matched. For example, a developer adds
additional statements to the body of an if-statement, the *Match* rules report
that the if-statement in the original version can be matched with the
corresponding if-statement in the modified version.

- *Convertibility* rules - identifies when one syntactic element can be
converted into another syntactic element of differing type. For example, a
developer changes a for-statement into a while-statement. Here, 

*Convertibility* rules report that the for-statement in the original version can be converted into the while-statement in the modified version.

- *Nesting* rules - identifies when a syntactic element can be placed within another. For example, a developer adds an if-statement as a guard around an existing statement. In this case, *Nesting* rules report that the existing statement in the original version can be nested within the if-statement in the modified version.

Each of these rule categories is now discussed separately and illustrated in the same manner as Figure 4.1 and Figure 4.2 (Unified View), with deleted text in red with a line-through (*deleted*), inserted text in a green/dark grey (*inserted*), and the original text left unchanged in normal font. The following is the result of srcDiff on the previous method example (as side-by-side diff).

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static boolean isStoragePolicyXAttr(XAttr xattr) {</td>
<td>public static String getStoragePolicyXAttrPrefixedName() {</td>
</tr>
<tr>
<td>return xattr != null &amp;&amp; xattr.getNameSpace() == XAttrNS</td>
<td>return XAttrHelper.getPrefixedName(XAttrNS, \</td>
</tr>
<tr>
<td>&amp;&amp; xattr.getName().equals(STORAGE_POLICY_XATTR_NAME); }</td>
<td>STORAGE_POLICY_XATTR_NAME);</td>
</tr>
</tbody>
</table>

The function *determineMatchings* from the srcDiff algorithm in Figure 4.3 utilizes the *Match* and *Convertibility* rules along with a pre-check belonging to the *Nesting* rules to determine if nesting a child is better than matching. The function *determineNestings* only utilizes the *Nesting* rules.
The differencing rules, as a whole, incorporate domain knowledge of how a developer determines what changed in a piece of code. These rules can be generalized to most any syntactic-differencing approach. The rules encode knowledge that is common across a number of programming languages. We originally developed the rules in the context of C++ but our underlying infrastructure, srcML, also supports Java. Only a small amount of effort (less than a day) was spent extending the rules to work for Java. In particular, we needed to support a number of Java specific constructs that are not present in C++ (e.g., static block).

4.3.1 Match Rules

Match rules are applied to two of the same syntactic elements, such as two declaration statements or two if-statements to determine if they are a match. More explicitly, the Match rules are used to determine if a) two elements are similar enough to break down into constitute parts and perform additional fine-grained differencing, or b) if it is better to produce a delta showing a replacement of the one element with the other. The goal here is to generate a delta that is easily understood by the developer. Currently, the Match rules are divided into five different types of actions as shown in Table 4.1.

The Always Match rule includes syntactic categories that will always be broken down, no matter how different. There are two reasons for this. The first is to break down small nodes, which typically contain only text (e.g., names, operators, etc.). In the following example, the only item changed between the original and modified code is the variable name. Instead of marking the entire identifier as deleted and a separate identifier inserted (i.e., the node and its text is deleted and a new node with separate text inserted),
the subparts of the name are marked as deleted and inserted (i.e., it is the same node, but the text is changed) to indicate a rename.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>KisPenWidget *m_optionsWidget;</td>
<td>KisPenWidget *m_options;</td>
</tr>
</tbody>
</table>

**Table 4.1. Match rules and the associated syntactic categories (srcML elements).**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Syntactic Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Always Match</strong></td>
<td>name (single identifier), type, then, condition, control, init, default, comment, block, private, protected, public, parameter_list, argument_list, member_init_list, argument, range, literal, operator, modifier</td>
</tr>
<tr>
<td><strong>Any Similarity Match</strong></td>
<td>expr, expr_stmt</td>
</tr>
<tr>
<td><strong>Name Match</strong></td>
<td>call, decl, decl_stmt, parameter, function, function_decl, constructor, constructor_decl, destructor, destructor_decl, class, struct, union, enum</td>
</tr>
<tr>
<td><strong>Condition Match</strong></td>
<td>while, switch, for</td>
</tr>
<tr>
<td><strong>If-Statement Match</strong></td>
<td>if</td>
</tr>
</tbody>
</table>

The second reason is when it does not make sense for a syntactic element to be repeated (e.g., an argument list). In the following example, there is no similarity between the original argument list and the modified argument list, besides the parenthesis. Thus, without the **Always Match** rule, the argument list will not be broken down, leaving the original argument list as deleted and the modified argument list as inserted. However, with this rule, the argument list is broken down and the inserted argument can be correctly marked.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>update();</td>
<td>update(true);</td>
</tr>
</tbody>
</table>

**srcDiff**

update(true);
The *Any Similarity Match* rule applies to expressions and expression statements, and is similar to *Always Match*, except if the two have no similarity at all or there is more deleted/inserted text than similar text. In both situations, a match does not occur. Otherwise, any amount of similarity constitutes a match. The elements in this rule are ubiquitous and can have very complex contents, which makes it valuable in many contexts not to perform finer-grained differencing thus obtaining more understandable deltas. In the following, there is no matching text between the two expressions, so the entire statement can be marked deleted and inserted.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b;</td>
<td>c - d;</td>
</tr>
<tr>
<td><strong>srcDiff</strong></td>
<td></td>
</tr>
<tr>
<td>a + b, c - d</td>
<td></td>
</tr>
</tbody>
</table>

In this next example, the expressions have some similarity without excessive dissimilarity. This rule, followed by the use of *Always Match*, has the operator marked as inserted and the identifier renamed.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b;</td>
<td>a - d;</td>
</tr>
<tr>
<td><strong>srcDiff</strong></td>
<td></td>
</tr>
<tr>
<td>a + b, c - d</td>
<td></td>
</tr>
</tbody>
</table>

The rule *Name Match* applies to elements which contain a name (e.g., functions and variable declarations). For *Name Match*, a matching name between both constitutes a match no matter what other differences exist. In its simplest form, a class in the original and modified code with the same name is considered the same, even if the contents are massively different. The following example contains a number of textual changes between
the original and modified code, such that the inserted declaration could be considered a
replacement if not for the matching variable name.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>QTextDocument * m_document;</code></td>
<td><code>QTextDocumentPtr m_document;</code></td>
</tr>
<tr>
<td><strong>srcDiff</strong></td>
<td></td>
</tr>
<tr>
<td><code>QTextDocumentPtr m_document;</code></td>
<td></td>
</tr>
</tbody>
</table>

A small extension is needed for method calls of an object, e.g., `str.size()` and the call to `setImage` in Figure 4.1. If the variable is a rename, then we still want a match to occur. `srcDiff` currently considers it a match if any of the names match.

The *Condition Match* rule applies to loop statements and the switch statement and produces a match if the conditional is the same. For instance, the following `while`-statement has a large number of changes to its block. However, as it still has the same condition, it is considered to be the same statement

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>while(args[0]) {</code></td>
<td><code>while(args[0]) {</code></td>
</tr>
<tr>
<td><code>  ++args;</code></td>
<td><code>if(strcmp(args[0], OUT_FLAG) == 0) {</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>  output_file = args[1];</code></td>
</tr>
<tr>
<td></td>
<td><code>  ++args;</code></td>
</tr>
<tr>
<td></td>
<td><code>  ++args;</code></td>
</tr>
<tr>
<td><strong>srcDiff</strong></td>
<td></td>
</tr>
<tr>
<td><code>while(args[0]) {</code></td>
<td><code>if(strcmp(args[0], OUT_FLAG) == 0) {</code></td>
</tr>
<tr>
<td><code>  if(strcmp(args[0], OUT_FLAG) == 0) {</code></td>
<td><code>  output_file = args[1];</code></td>
</tr>
<tr>
<td><code>  ++args;</code></td>
<td><code>  ++args;</code></td>
</tr>
<tr>
<td>`}</td>
<td>`}</td>
</tr>
</tbody>
</table>

In the case of a for-statement, the initialization and increment are considered in addition to the condition (i.e., the whole control group). In the following, the contents of
the block are completely different. However, since the control group is the same, it is treated as the same for-statement. In contrast, if any part of the control group does not match, then it will not pass this rule.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>for(i=n.begin(); i!=n.end(); ++i) { std::cout &lt;&lt; i-&gt;treeID() &lt;&lt; '\n'; }</td>
<td>for(i=n.begin(); i!=n.end(); ++i) { int id = i-&gt;treeID(); if(id == treeID) return *i; }</td>
</tr>
</tbody>
</table>

A special, and more complex, rule is applied to if statements, the *If-Statement Match*. The simplest situation is when both if-statements have identical then-clauses. In this case, they are considered the same if-statement. The following is an example:

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(p &gt;= count()) break;</td>
<td>if(already_exist(chain, p)) break;</td>
</tr>
</tbody>
</table>

The conditions of the two if-statements have very little similarity and the amount of change between the statements is greater than the similarity. However, since the then-clauses are identical, it is considered the same if-statement.

The other rules associated with an if-statement are related to when both conditions are identical. First, the rule produces a match if either of the following is true about the original and modified if-statements: 1) both contain a block-statement ({ }) in the then-clause; or 2) both do not have a block-statement in the then-clause. In the following, both
if-statements have the same condition, but very different statements in the block. Despite this, they should be considered the same if-statement.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(name == null) {</td>
<td>if(name == null) {</td>
</tr>
<tr>
<td>return false;</td>
<td>throw new IllegalException(&quot;...&quot;);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

Second, the rule produces a match when any of the following is true about the original and modified if-statements: 1) both contain an else-clause; 2) both do not contain an else-clause; or 3) only one has a block-statement, while the other if-statement does not have an else-clause. This complexity is needed in cases such as when an if-statement is taken from around code and an else-clause added. The following two examples illustrate the condition/block/else situation. In the first example, the modified if-statement has a block and the other does not. The if-statement appears to be a previous version of the other, so matching is expected.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(!paraStyles.contains(root.ID()))</td>
<td>if(!paraStyles.contains(root.ID())) {</td>
</tr>
<tr>
<td>tree_root &lt;&lt; root.ID();</td>
<td>int id = root.ID();</td>
</tr>
<tr>
<td>}</td>
<td>if(id != tree_rootID)</td>
</tr>
<tr>
<td></td>
<td>tree_root.insert(id);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>srcDiff</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>return false;</td>
<td>throw new IllegalException(&quot;...&quot;);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>srcDiff</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>srcDiff</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>srcDiff</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In the next example, the original if-statement has a block and the other does not. Here, they do not match so the second modified if-statement can be placed in the original if-statement. The main difference is the addition of an else-clause. With the additional else-clause logic, the correct delta is achieved.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (shape.has(&quot;anchor&quot;)) {</td>
<td>string type;</td>
</tr>
<tr>
<td>string type = shape.get(&quot;anchor&quot;);</td>
<td>if (shape.has(&quot;anchor&quot;))</td>
</tr>
<tr>
<td>if(type != &quot;page&quot;) {</td>
<td>type = shape.get(&quot;anchor&quot;);</td>
</tr>
<tr>
<td>// several lines common code</td>
<td>else</td>
</tr>
<tr>
<td>}</td>
<td>type = &quot;as-char&quot;;</td>
</tr>
<tr>
<td></td>
<td>if(type != &quot;page&quot;) {</td>
</tr>
<tr>
<td></td>
<td>// several lines common code</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

This rule is derived from examining the change history of existing projects and fails when the extracted if-statement has no else-clause. However, if an else-clause is not needed, then it does not need extraction.

For code that does not pass one of the rules above, there are two fallback rules applied in succession: syntactic similarity and textual similarity. These are also used as part of Convertibility and Nesting rules. Both rules use similarity and dissimilarity metrics computed using Myers’ Shortest Edit Script Algorithm [Myers 1986]. Unlike Levenshtein distance [Levenshtein 1966], substitutions are not a single operation, but a delete and an
insert. We used Myers’ algorithm as substitutions are not required by the metrics. These two metrics are defined as:

\[
\text{similarity} = \min(#\text{original} - #\text{deletions}, #\text{modified} - #\text{inserts})
\]
\[
\text{dissimilarity} = #\text{deletions} + #\text{inserts}
\]

where \(\#\text{original}\) and \(\#\text{modified}\) is the number of textual tokens in the case of \textit{textual similarity}, and the number of child elements in the case of \textit{syntactic similarity}. The \#\text{deletions} and \#\text{inserts} are the number of inserted and deleted lexical tokens/child elements.

\textit{Syntactic similarity} is computed on the child elements of two candidate elements after stripping out immediate child literals, modifiers, operators, empty argument lists, and text tokens (e.g., the if keyword when applied to an if-statement). If similarity is greater or equal than half the minimum number of children between both sets and the amount of dissimilarity is not greater than this minimum, then the two candidate elements are a match. Depending on how the AST is formed, the syntactic-similarity rule can subsume the \textit{Condition Match} rule (i.e., a for, while, or switch consist only of control group/condition and block statements). In these instances, the \textit{Condition Match} rule may be considered a quick pre-check to avoid a costlier rule. Additionally, if both elements have a block as the final child or an if-statement with only a then-clause, and the previous check fails, the same syntactic check is also applied on children of the block.

The final fallback rule is \textit{textual similarity} (Figure 4.5). With exception of rule 2 (Figure 4.5), it is computed on the textual leaf tokens after stripping out non-operator
punctuating characters such as (, ), {, }, [, ], :, ;, and , and operators/modifiers part of a name (except : :). Rule 2 uses measures without stripping out any tokens.

\[
\begin{align*}
\text{minSize} &= \min(#\text{originalTokens}, #\text{modifiedTokens}) \\
\text{maxSize} &= \max(#\text{originalTokens}, #\text{modifiedTokens})
\end{align*}
\]

\begin{verbatim}
1 if ((similarity == minSize) && (dissimilarity < 2*minSize)) return true;
2 if (similarity_no_strip == min_size_no_strip) return true;
3 if (dissimilarity > maxSize) return false;
4 if ((minSize < 30) && (dissimilarity > 1.25*minSize)) return false;
5 if ((minSize ≥ 30) && (dissimilarity > 1.05*minSize)) return false;
6 if (minSize ≤ 2) return similarity ≥ ½* minSize;
7 else if (minSize ≤ 3) return similarity ≥ 2/3* minSize;
8 else if (minSize ≤ 30) return similarity ≥ 7/10*minSize;
9 else return similarity ≥ ½* minSize;
\end{verbatim}

Figure 4.5. Textual similarity rules. The rules use similarity, dissimilarity and size measures to test if two elements are similar enough textually. The measures, with exception to rule 2, are calculated after stripping out punctuation characters and other tokens that negatively affect performance. Rule 2 uses all tokens to test for an exact match. It is not applied to Nesting rules.

The first two if-statements (rules 1 and 2) handle complete matches (i.e. one child is a complete subset). The first applies to all the rules, while the second does not apply to Nesting rules. The 3rd - 5th rules encode that a potential match should not have an excessive amount of dissimilarity. The remaining rules constitute the main textual-similarity tests. When the list is two or three tokens, a majority of them need to be the same. Finally, it was identified in our empirical analysis of changes that a relatively small structure (e.g., <30 tokens) needs a larger amount of similarity to be considered a match, while a larger structure can be a simple majority. That is, an if-statement of two or three lines (smaller number of tokens) requires a closer match than an if-statement of say 50 lines (large number of tokens).
4.3.2 Convertibility Rules

Convertibility rules occur between two syntactic elements of different types and determines if one of the elements can be converted to the other, such as converting a for-statement to a while-statement. When identified, srcDiff marks the elements with delete and insert tags each with attribute type and value convert and performs fine-grained differencing on their children via recursively applying the srcDiff algorithm on the children. There are currently five types as listed in Table 4.2.

Table 4.2. Convertibility Rules organized by type. Each type has the associated syntactic categories.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Syntactic Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>class, struct, union, enum</td>
</tr>
<tr>
<td>Access Regions</td>
<td>private, protected, public</td>
</tr>
<tr>
<td>Logical</td>
<td>if, while, for</td>
</tr>
<tr>
<td>Else</td>
<td>else, elseif</td>
</tr>
<tr>
<td>Statement</td>
<td>expr_stmt, decl_stmt, return</td>
</tr>
</tbody>
</table>

All the convertible rules use the Match fallback rules as a check to see if one can be converted to the other. Class, Logical, and Statement have additional conditions that will result in passing. For the Class rule, a conversion can occur if they both have the same name. The Logical rule passes if both have the same condition (similar to the Condition Match, except only the condition of a for is used). The following is an example of a for-statement converted to a while-statement.
## srcDiff

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>int itemCount = d.items.count(); for(int i = itemCount-1; i &gt;= 0; --i) { Item &amp;sbItem = d.items[i]; if (sbItems.widget() == widget) { // several common lines } }</td>
<td>int itemCount = d.items.count() - 1; while(i &gt;= 0) { Item &amp;sbItem = d.items[i]; if (sbItems.widget() == widget) { // several common lines } --i; }</td>
</tr>
</tbody>
</table>

For **Statement**, the top-level expression in each is used to determine if they can be converted, i.e., if the expressions can be matched (initialization in declaration-statement, returned expression in return-statement, and the complete expression in expression-statement). To be converted, they both must have expressions, the expressions must pass the **Match** fallback rules, text similarity must be greater than half the size of the largest expression, and text dissimilarity must be less than half the size of the largest expression. In the case of matching a declaration statement to one of the other, then the size of the expression trees must also not be greater than four times each other. The following is a simple, but real change example.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool ok = openUrl(url);</td>
<td>openUrl(url);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>srcDiff</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool ok = openUrl(url);</td>
</tr>
</tbody>
</table>
4.3.3 Nesting Rules

Nesting rules occur between syntactic elements with the goal to determine if one or more of the elements can be safely nested (placed) within another. That is, mark the one element as deleted/inserted and recursively apply the srcDiff algorithm on the nest-able elements and that element's children to determine if they can be matched or further nesting is required. For example, an if-statement is deleted, however, the statements in the if-statement’s body are left unchanged. The nesting rules determine that the statements in the modified document can be nested within the deleted if-statement where they can later be marked as unchanged.

The Nesting rules are applied to unmatched elements, i.e., those that failed Match/Convertibility rules. However, Nesting rules are also checked during the Match/Convertibility process to see if an element can and will create a better difference by nesting it into the potential match/conversion or a subsequent element. If so, the match/conversion is rejected to allow for the nesting. An element is better nested with its best internally matching element when it has greater or equal text similarity and less than or equal text dissimilarity, or it has a better ratio of minimal text node size to text similarity and passes a slightly modified version of Match rules. This is described in more detail in following paragraphs.

We constructed rules for syntactic elements (statement and sub-statement) that can contain nested elements. For example, an if-statement can have other if-statements, blocks, while loops, etc. Unmatched elements are checked to see if they meet the conditions for nesting. Once identified, additional checks are made. First, the best internally matching
element (same element, both no greater than four times the size of the other, and highest text similarity) is found. In the case of a node having descendants with the same matching node name, only the ancestor is taken (a subsequent nesting will nest further if required). If no item is found, then that item will not be nested. Then, they are checked using rules similar to the *Match* rules except, then-clause, block, comment, literal, operator, modifier, expression, expression-statement, and name are no longer *Always Matched* or *Any Similarity Match*, but use the fallback rules. The remaining details are the same as the *Match* rules.

If the elements have the same type (i.e., two if-statements), an additional check is made (both when determining if better to nest or if can be nested). The best internally matching element must have greater or equal text similarity and less than or equal amount of text dissimilarity (i.e., it is more optimal to nest), or the smallest element (in terms of textual nodes) must be sufficiently large ($\geq 50$), the ratio of minimum size of text nodes to text similarity must be less than 90% when nested as compared to not being nested, and they must pass the previously discussed modified Match rules for nesting. The second condition catches false negatives. The best nesting may have slightly less total similarity (due to other changes) and still produce a more meaningful difference. However, to reduce false positives, it must have a sufficiently better ratio and is bounded to work for larger structures (we picked $\geq 50$ after tuning) where ratios are more stable. The following is an example with the $\geq 50$ relaxed due to space restrictions. In this example, the parent if-statement was deleted and its child declaration statement moved into the child if-statement. Because of the move of the declaration statement there is less similarity between the child
if-statement in the original and the matching modified if-statement. However, since the original child if-statement is smaller the ratio is better by a notable margin.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
</table>
| if(root.ID()) {  
  node left = root.left;  
  if(child.ID() < root.ID()) {  
    this->insert(left, child);  
  }  
} | if(!root.ID()) return;  
if(!root.ID()) return;  
if(child.ID() < root.ID()) {  
  node left = root.left;  
  this->insert(left, child);  
} |

Next, checks are made to see if a better nesting is possible. These are (for nesting into a deleted item): in the reverse direction (deleted element within the inserted), the inserted element in the next deleted element, and deleted element inside the next inserted element. The checks are similar for nesting into an inserted element. Lastly, there are some special checks for identifiers. First, if the parent of both the identifier and what it may potentially be nested into are expressions, then there must only be one inserted element and deleted element to check for nesting. Secondly, the identifiers (one being nested and the potential match when nested) either must have parents with the same element type or an identifier in an expression is to be nested within a function call identifier i.e., foo -> foo.bar(). In the latter case, prefixes of the identifier sequence and call name identifier sequence have to be sufficiently similar (at least half the longest identifier sequence length).
The following is an example where the modified code introduced an if-statement that wraps the statement from the original. In this example, the delete-statement is iteratively nested through recursive application of the srcDiff algorithm. The initial application places the delete-statement in the if-statement, a second application places it in the then-clause, and a final recursive application placed the delete-statement in the block.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
</table>
| delete m_document; | if (m_frames.isEmpty()) {
  delete m_document;
} |

`srcDiff`

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
</table>
| if (m_frames.isEmpty()) {
  delete m_document;
} | if (m_frames.isEmpty()) {
  delete m_document;
} |

The following is an example of a nest where the element being nested and the element being nested into are the same elements. Here, the original if statement is removed and the statements in the block are kept. Subsequently, the statements in the modified code are nested into the if statement in the original.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
</table>
| if (shape.has("anchor")) {
  string type = shape.get("anchor");
  if (type != "page") {
    Anchor *anchor = new Anchor(shape);
    anchor->loadShape(element);
  }
} | string type = shape.get("anchor");
if (type != "page") {
  Anchor *anchor = new Anchor(shape);
  anchor->loadShape(element);
} |

`srcDiff`

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
</table>
| if (shape.has("anchor")) {
  string type = shape.get("anchor");
  if (type != "page") {
    Anchor *anchor = new Anchor(shape);
    anchor->loadShape(element);
  }
} | string type = shape.get("anchor");
if (type != "page") {
  Anchor *anchor = new Anchor(shape);
  anchor->loadShape(element);
} |
4.4 Complexity

The following present the worst-case time complexity of the srcDiff algorithm. For illustration of its practical purposes, a timing comparison is performed as part of the evaluation in CHAPTER 7. Let N be the maximum number of children for a node, D be the maximum number of individual edits (inserts and deletes\(^3\)) between children (which is always less than or equal to 2N), C be the cost of comparing two child nodes\(^4\), and R be the cost of the srcDiff rule computation. Myers’ algorithm (Line 2) then runs in 2CND or O(CND) time. In Line 3, the D edits are checked for identical matches of the opposite edit type using a hash map. In the worst case, when everything hashes to the same value and using child node comparison to disambiguate, this is O(CD\(^2\)) (which is less than CND). Line 4 iterates through the groupings of consecutive edits, consecutive common children, and changes, so at most O(N). Common, deleted, and inserted children are output (lines 5-8), which over all recursive calls to the srcDiff algorithm and in combination with lines 13, 19, and 22, results in the output of all nodes. Let M be the maximum number of nodes between both trees, so output takes at most 2M or O(M). Line 10 uses a quadratic dynamic programming algorithm. At worst all D edits will belong to a single change or O(RD\(^2\)) (in

\(^3\) For D, deletes and inserts comprising a change are counted and not the change itself.

\(^4\) As a practical note, a hash is computed on demand for each node set minimizing the cost of successive comparisons. This was added as a refinement after the evaluation for both comparisons and markMoves (Line 3).
this case Line 10 will be \( O(1) \)). Line 16 (at worst when there are no matches) checks if each node can be nested in the other for \( O(RD^2) \) (this only occurs when it finds nothing that can be nested). Combined, these will result in at worst \( D \) recursive calls. Finally, let \( K \) be the maximum depth of the srcML tree. Lines 10 and 16 will result in \( O(D^K) \) total recursive calls. So, the runtime complexity of the srcDiff algorithm is \( O(D^K(CND + RD^2) + M) \). In practice, \( K, N, \) and \( D \) are not typically large numbers. In addition, recursion is stopped (recursive subtrees pruned) when children are determined to be common, deleted, or inserted. Since, a large part of the srcDiff rules rely on children being similar enough, excessively dissimilar children will not result in a recursive call and thus, expected runtime is much less.

For the srcDiff rules (i.e, \( R \)), textual similarity and syntactic similarity are the costliest as they both employ Myers’ algorithm on the text nodes and child nodes, respectively. The two times these are used most heavily (as part of the rules) is when checks are made in determineMatchings to see if it would be better to nest a child in one of the subsequent others of the other edit type and when searching for the best (highest similarity) descendant of a certain type. In the first case, textual similarity and syntactic similarity might be applied \( O(D) \) times. In the latter, textual similarity will be applied once to each found descendant. Of note, the srcDiff rules are not run on every child part of an edit. Match rules will never be applied to two elements of separate type, so, \( R \) is constant for those children. Similarly, Convertibility rules and Nesting rules first check if an element can be converted to or nested in the other through simple lookups.
In practice, from various experiments on applying the approach to the complete history of software systems, the bottleneck is primarily textual similarity (and other Myer’s shortest edit script usages). The only instance this was a problem was in the case of extremely large auto-generated files. For this purpose, approximation was added in place of Myer’s algorithm when one of the sequences is greater than 2048.
CHAPTER 5

srcDiff Tool

In this chapter, we discuss the current implementation of the srcDiff tool as well as the interface for using the tool. The tool is called srcdiff.

5.1 Implementation

srcdiff is implemented in C++ and uses CMake [Kitware] as a build system. In total, srcdiff consists of over 16 KLOC distributed among the following main modules: client, translator, srcml, shortest_edit_script, and view. Figure 5.1 shows a simplified high-level UML Class diagram of the modules.

Figure 5.1. A simplified high-level view of modules in srcdiff.
The client module is responsible for handling user input (command-line interface) and managing and supporting various input sources. Currently, local filesystem, Git repositories, and limited support for Subversion repositories are supported as input sources. The parsing of the command-line arguments is handled through boost’s [Dawes, Abrahams, Rivera 1998-2007] program_options library [Prus 2002-2004]. As part of developing srcdiff, a fault was found and a patch submitted to boost’s program_options library where it was eventually included as part of the 1.59.0 release of boost. Once the options are parsed, the input source is processed and sequences of input file pairs are passed to the translator module.

The translator module is responsible for implementing the srcDiff approach and rules. That is, it computes the difference between the two supplied inputs. Once the translator module is invoked, each of the inputs convert themselves to srcML via the srcml module. The conversion of both original and modified input can be done in parallel via threads\(^5\). In addition to providing conversion of input to srcML, the srcml module is also responsible for traversing the srcML and generating the format used internally by srcdiff. The srcml module converts the input to srcML using libsrcml (CHAPTER 3). The srcML generated by libsrcml is then processed using libxml2’s Text Reader interface [Veillard 2004a] into a sequence of srcML nodes, one for each input.

\(^5\) Currently, some part of srcdiff or libsrcml is not thread safe. Once fixed, srcdiff can support srcML conversion in parallel with threads.
The translator module forms node sets (child sub-trees) on demand from these sequences. For example, each child of root is converted into a node set, which consists of all the nodes between the opening and closing srcML tags for that child. When required, such as when recursively calling the srcDiff algorithm on two matched children, node sets for those children’s children are created. The current implementation of the node sets does not copy any srcML nodes, but consists of a list of position offsets into the srcML node sequences.

After the srcML nodes are generated and the first node sets created, translator module starts the srcDiff algorithm (Figure 4.3). The first part of the srcDiff algorithm, computeShortestEditScript (the execution of Myers’ shortest edit script algorithm [Myers 1986]), is implemented via the shortest_edit_script module. The shortest_edit_script module implements a custom implementation of Myers’ shortest edit script algorithm.

In [Myers 1986], Myers’ gives several versions of the same algorithm supporting separate tradeoffs. The shortest_edit_script module implements two of these: the normal O(ND) space complexity and the linear-space refinement version. The O(ND) version executes more quickly, while the linear-space version scales to larger sequences. The shortest_edit_script module combines the two into a hybrid version. The linear-space version works by repeated execution with each execution identifying a middle that is always part of the final solution. That is, it splits the sequences into two subparts with each execution. The hybrid version selects the linear-space version for large sequences and switches to the O(ND) version when the size becomes small enough. The
first call to the hybrid version uses a best guess based on the length of the sequences to
decide which version to run, however, each call to the linear-space version reports the total-
edit distance which can be used to accurately gauge the correct point to switch to the O(ND)
version.

Once the shortest-edit script is computed, the translator module applies the remainder of the srcDiff algorithm. During the execution of the srcDiff, once a node has been determined to be common, inserted, or deleted, it is output. That is, the final srcDiff is output incrementally through the execution of the algorithm. This is accomplished directly via libsrcml, which provides support for the incremental creation of a srcML (in this case srcDiff) document in memory. Once the srcDiff algorithm is finished, the complete srcDiff for the input pair is contained in memory.

Once the srcDiff for an input pair is created, the srcDiff is either appended to the srcDiff archive being created (when the desired output is the srcDiff format) or the view module is invoked to transform the srcDiff output into the desired view such as the human-readable unified or side-by-side views. After the evaluations were completed, the view module was extended to provide customizable output themes and various levels of syntax highlighting. Figure 5.2 shows sample output of the view module on the source-code from Figure 4.1 and Figure 4.2 (the source code is also reproduced in the top of Figure 5.2). The first view is srcdiff’s default theme and default level of syntax highlighting. The second view is srcdiff’s Monokai theme (based off of Sublime Text’s [Ltd 2016] Monokai theme). Both of these outputs are produced as HTML by srcDiff and rendered
via a browser. The default theme is also supported as output to the terminal. In addition, the view module supports user-defined themes.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void Settings::setImage( KisImageSP image ) {</code></td>
<td><code>void Settings::setImage( KisImageSP image ) {</code></td>
</tr>
<tr>
<td><code>m_widget-&gt;m_filter-&gt;setImage( image );</code></td>
<td><code>if (m_options) {</code></td>
</tr>
<tr>
<td></td>
<td><code>m_options-&gt;m_filter-&gt;setImage( image );</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

**Default Theme and Default Syntax Highlighting**

```cpp
@@ -1 +1 @@
void Settings::setImage( KisImageSP image ) {
    if (m_options) {
        m_widget->m_options->m_filter->setImage( image );
    }
}
```

**Monokai Theme and Full Syntax Highlighting**

```cpp
@@ -1 +1 @@
void Settings::setImage( KisImageSP image ) {
    if (m_options) {
        m_widget->m_options->m_filter->setImage( image );
    }
}
```

![srcDiff output examples](image)

**Figure 5.2.** Examples of srcDiff output provided by the view module. Top shows the original and modified source code, middle is the default theme and default level of syntax highlighting. Bottom is the Monokai Theme with full syntax highlighting. Both outputs are taken from srcDiff’s view module to HTML.

After the view is generated, the process is repeated for each input pair contained in the input sources.

### 5.2 Interface

`srcdiff` is a command-line interface based off of srcml’s command-line interface. As such it accepts many of the same arguments. One of the main differences is
that it requires sequences of file pairs (or directory pairs) with one being the original file (or directory) and the other being the modified file (or directory). The following is the most basic way to run srcdiff. It takes the two files: original.cpp and modified.cpp and produces output to the terminal in the srcDiff format.

```
srccdiff original.cpp modified.cpp
```

The following does the same, except it places the output into the file srcdiff.xml.

```
srccdiff original.cpp modified.cpp -o srcdiff.xml
```

It can also take two directories and produces a srcDiff archive containing the differences of all the files in those directories. As a note, this requires matching names and directory structures to work as expected. Here all the code in the directories original and modified is differenced and placed in the srcDiff archive srcdiff.xml.

```
srccdiff original/ modified/ -o srcdiff.xml
```

srcdiff can also take a file containing a list of files to difference (i.e., a file list). The file list consists of file pairs. Each file pair is on a single line and of the form: original_file_and_path|modified_file_and_path. With this option, the files can be in separate locations (non-matching directory structures). Here is an example
where the files list, file_list.txt, is read and each file pair is differenced and placed in the srcDiff archive srcdiff.xml.

    srcdiff --files-from file_list.txt -o srcdiff.xml

srcdiff supports separate input sources other than the local file system. It can take input from Git and Subversion repositories (Subversion implementation currently does not support authentication). The following are examples. Both take the URL of the repository, url, and the two revisions to compute the difference of: original and modified. With Git, --git is used, and with Subversion, --svn is used. The output is a srcDiff archive containing the delta of the two revisions. These options can also be combined with the --files-from option to only difference the specified files.

    srcdiff --git url@original-modified -o srcdiff.xml
    srcdiff --svn url@original-modified -o srcdiff.xml

Human-readable outputs are specified with --unified (or -u) for the unified view or --side-by-side (or -y) for the side-by-side view. Both can be configured to display a different number of lines of common context code. The unified view also supports additional context options: all for all code, function to display the entirety of a function with a change. The following will produce the unified view to the terminal for original.cpp and modified.cpp with a default of three lines of context.
srcdiff -u original.cpp modified.cpp

The following does the same with a hundred lines of context and all context respectively.

srcdiff -u100 original.cpp modified.cpp
srcdiff -uall original.cpp modified.cpp

To output HTML instead the --html switches to output html for viewing in a browser.

srcdiff -u --html original.cpp modified.cpp

srcdiff can be set to not report changes to whitespace with the --ignore-space (-w) or the --ignore-all-space (-W) options and comments with --ignore-comments (-c) option. Option -W will not report any whitespace changes. However, this can break up consecutive inserts and deletes of the same type. So, the -w option does not break up consecutive code inserts and deletes by marking up whitespace in between. The following produces the unified view, but does not mark up whitespace and comments.

srcdiff -u -w -c original.cpp modified.cpp
Coloring themes are supported via the `--theme` option. The themes can be default, `monokai`, or a file containing a user-defined theme. In addition, the level of syntax highlighting can be specified with `--highlight`. Syntax highlighting can be set to `none`, `partial` (default), or `full`. The following will produce the unified view in HTML with the user specified theme in `theme.txt` and full syntax highlighting.

```
srcdiff -u --theme=theme.txt --highlight=full o.cpp m.cpp
```

As a command-line tool, `srcdiff` can be easily integrated with Version Control Systems such as Git. APPENDIX A contains the code for a wrapper to integrate `srcdiff` with a Git repository.

A complete list of all the options `srcdiff` currently supports is available in APPENDIX B.
CHAPTER 6

Preliminary Study

In order to evaluate the rules, we first performed a preliminary study comparing the output of srcDiff to the widely used GNU diff.

For the preliminary study, we choose to compare our tool with the standard version of GNU diff for several reasons. It is the de-facto standard of diff tools and it, or the underlying line-differencing algorithm, is used by millions of developers. A comparison with it is one obvious baseline. The full study compares srcDiff to additional textual approaches that support coloring of changes and within-line highlighting.

We did not compare srcDiff to a syntactic differencer as these approaches are not commonly used or adopted. Furthermore, very few provide capability of outputting the changes in a visual format (and thus comparable to srcDiff). GumTree is a more recent syntactic differencing that provides a visual output and thus can be compared to srcDiff. GumTree is also compared with srcDiff as part of the full study.

An online survey (utilizing Qualtrics) was conducted comparing the deltas produced by srcDiff to those produced by GNU diff. A PDF of the survey is included in a replication package (sdml.cs.kent.edu/srcDiff/dissertation/preliminary_study). The survey consisted of a pre-questionnaire, a sample problem to familiarize the participants with the format and questions, twelve change samples given in a random order, and an exit questionnaire.
The samples were collected from a corpus of over 1,000 bug-fix revisions on KOffice through a combination of manual inspection, mining for the most common syntactic changes, and changes conforming to Pan’s Bug-fix pattern taxonomy [Pan, Kim, Whitehead 2009]. The samples contained both the lines that changed and context if necessary. From these methods, 77 different changes were collected. Of these, 12 complicated changes with a mix of insertions and deletions were selected to exercise the heuristics. Changes with only deletes or only inserts are generally easy to understand. The mixed cases we selected are fairly difficult to understand (and the focus of this study). The samples range from single lines to complete functions. The example given to participants and all samples are provided in APPENDIX C in the same order as presented here. In addition, the PDF of the survey in the replication package contains the samples, also in the same order as presented here.

For the survey, all tasks were timed, all questions were required to be answered for a given change before moving on to the next, and participants were not permitted to return to view a previous change and their answers.

For each change, the participants are presented with the original and modified source code, the GNU diff view of the source-code changes, and the srcDiff view of the source-code changes (all automatically generated from source code). For the GNU diff view, the participants were provided with complete unified context (unified context is the default mode used for version control systems such as Subversion and Git).

For srcDiff, a SAX (Simple API for XML) program was used to transform the XML output into a more easily read format (this study pre-dated the currently provided view
output). For simplicity this shall be denoted the srcDiff view. Code that was changed between the versions was given the background color red and marked with a line-through, inserted code was given the background color green, and code in common to both versions was left as is. In addition, to be able to see where newlines were inserted and deleted the Unicode carriage-return symbol was inserted before the newline when part of a change. Any amount of context (i.e., lines or entire function containing the change) can be given in the srcDiff view however, to be comparable to GNU diff all lines of context were provided. Both programs were run to treat whitespace changes as significant. The replication package contains all the questions presented in the study.

For each change, the following questions were asked on a five-point Likert scale (codified as 6-10). First, the questions were asked for the GNU diff and then, the identical questions were asked for the srcDiff view.

- **Q1:** Some code marked as changed would be better marked as common.
- **Q2:** Some code marked as common would be better marked as changed.
- **Q3:** Does an excellent job of capturing the differences between the original and modified code.
- **Q4:** Presents differences in a manner that is easy to understand.

Following Likert questions, participants were asked to rank the Original and Modified, GNU diff, and srcDiff view in the order in which they preferred each view (Q5). After answering the questions for the twelve changes, the participants were given an exit questionnaire that asked the same questions about the GNU diff and the srcDiff views overall. In addition, the participants were asked to give any additional comments.
Students in Kent State University’s undergraduate Software Engineering course and The University of Akron’s Graduate Software Engineering Methodology course were given the survey to take as a homework assignment. In addition, students from Youngstown University’s Software Engineering course were given the option to take the survey for extra credit. In total, there were 131 participants and 97 completed the survey (74% completion).

6.1 Hypotheses

In order to evaluate srcDiff and the heuristics, we formulated the following hypotheses.

- **Hypothesis 1**: srcDiff produces a view of source code that is more accurate than GNU diff.
- **Hypothesis 2**: srcDiff produces a view of source code changes that is easier to understand than GNU diff.
- **Hypothesis 3**: srcDiff provides a view of source code changes that is more preferable than GNU diff.

In order to provide support for our hypotheses, we formulated the following null and alternate hypothesis based on Q1-Q4 from the survey (Table 6.1).
Table 6.1. List of null and alternate hypothesis tested for questions: Q1-Q4.

<table>
<thead>
<tr>
<th>Hypothesis 1</th>
<th>$H_{10}$</th>
<th>There is no difference between GNU diff and srcDiff in marking code as changed that should be marked as common.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$H_{1a}$</td>
<td>There is a difference between GNU diff and srcDiff in marking code as changed that should be marked as common.</td>
</tr>
<tr>
<td></td>
<td>$H_{20}$</td>
<td>There is no difference between GNU diff and srcDiff in marking code as common that should be marked as changed.</td>
</tr>
<tr>
<td></td>
<td>$H_{2a}$</td>
<td>There is a difference between GNU diff and srcDiff in marking code as common that should be marked as changed.</td>
</tr>
<tr>
<td></td>
<td>$H_{30}$</td>
<td>There is no difference in how excellent a job GNU diff and srcDiff markup differences.</td>
</tr>
<tr>
<td></td>
<td>$H_{3a}$</td>
<td>There is a difference in how excellent a job GNU diff and srcDiff markup differences.</td>
</tr>
<tr>
<td>Hypothesis 2</td>
<td>$H_{40}$</td>
<td>There is no difference in how easy GNU diff and srcDiff views are to understand.</td>
</tr>
<tr>
<td></td>
<td>$H_{4a}$</td>
<td>There is a difference in how easy GNU diff and srcDiff views are to understand.</td>
</tr>
</tbody>
</table>

For **Hypothesis 1**, a rejection of all three null hypotheses: $H_{10}$, $H_{20}$, $H_{30}$ with $H_{10}$ and $H_{20}$ having a smaller mean and $H_{30}$ having a larger mean would be ideal and would indicate that srcDiff is perceived to incorrectly mark less code ($H_1$ and $H_2$) and do a more excellent job at marking up the code ($H_3$). However, rejection of any hypothesis in a positive direction (and none in a negative direction) will provide support for **Hypothesis 1**. For **Hypothesis 2**, a rejection of the $H_{40}$ with a larger mean will provide support that srcDiff is perceived to be easier to understand.

### 6.2 Results

All data was analyzed using SPSS. Responses to Q1-Q4 were tested for reliability using Cronbach’s alpha. All had good ($\geq 0.7$) to excellent reliability ($\geq 0.9$). A two-tailed
paired T-test (applicable under Central Limit Theorem) with a 95% confidence interval was run between the GNU diff and srcDiff view responses independently for Q1, Q2, Q3, and Q4 and for each sample as well as the exit questionnaire for a total of 52 tests (Table 6.2 and Table 6.3). Information about effect size (i.e., Cohen’s d) is in Table 6.4 and Table 6.5.

Table 6.2. Results of Two-tailed Paired T-Tests on Q1 and Q2. Left-most column is the sample, and top-most is the questions. For each question, the mean of GNU diff and srcDiff views are given along with the t-test p-value and finally if we can reject the null hypothesis.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Accuracy Changed (Q1)</th>
<th>Accuracy Common (Q2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>diff µ</td>
<td>srcDiff µ</td>
</tr>
<tr>
<td>1</td>
<td>7.88</td>
<td>7.34</td>
</tr>
<tr>
<td>2</td>
<td>7.9</td>
<td>7.25</td>
</tr>
<tr>
<td>3</td>
<td>7.9</td>
<td>7.7</td>
</tr>
<tr>
<td>4</td>
<td>7.97</td>
<td>7.58</td>
</tr>
<tr>
<td>5</td>
<td>7.92</td>
<td>7.46</td>
</tr>
<tr>
<td>6</td>
<td>8.28</td>
<td>7.73</td>
</tr>
<tr>
<td>7</td>
<td>7.53</td>
<td>7.45</td>
</tr>
<tr>
<td>8</td>
<td>8.06</td>
<td>7.7</td>
</tr>
<tr>
<td>9</td>
<td>8.15</td>
<td>7.82</td>
</tr>
<tr>
<td>10</td>
<td>7.79</td>
<td>7.57</td>
</tr>
<tr>
<td>11</td>
<td>7.8</td>
<td>7.53</td>
</tr>
<tr>
<td>12</td>
<td>7.91</td>
<td>7.34</td>
</tr>
<tr>
<td>Exit Questionnaire</td>
<td>8.28</td>
<td>7.54</td>
</tr>
</tbody>
</table>
Table 6.3. Results of Two-tailed Paired T-Tests on Q3-Q4. Left-most column is the sample, and top-most is the questions. For each question, the mean of GNU diff and srcDiff views are given along with the t-test p-value and finally if we can reject the null hypothesis.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Accuracy (Q3)</th>
<th>Understandability (Q4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>diff&lt;sub&gt;µ&lt;/sub&gt;</td>
<td>srcDiff&lt;sub&gt;µ&lt;/sub&gt;</td>
</tr>
<tr>
<td>1</td>
<td>8.38</td>
<td>9.01</td>
</tr>
<tr>
<td>2</td>
<td>8.58</td>
<td>9.16</td>
</tr>
<tr>
<td>3</td>
<td>8.49</td>
<td>9.19</td>
</tr>
<tr>
<td>4</td>
<td>8.37</td>
<td>9.05</td>
</tr>
<tr>
<td>5</td>
<td>8.41</td>
<td>9.11</td>
</tr>
<tr>
<td>6</td>
<td>8.35</td>
<td>8.99</td>
</tr>
<tr>
<td>7</td>
<td>8.42</td>
<td>9.01</td>
</tr>
<tr>
<td>8</td>
<td>8.26</td>
<td>8.94</td>
</tr>
<tr>
<td>9</td>
<td>8.02</td>
<td>8.99</td>
</tr>
<tr>
<td>10</td>
<td>8.35</td>
<td>8.89</td>
</tr>
<tr>
<td>11</td>
<td>8.16</td>
<td>9.06</td>
</tr>
<tr>
<td>12</td>
<td>8.49</td>
<td>9.06</td>
</tr>
<tr>
<td>Exit Questionnaire</td>
<td>8.49</td>
<td>9.26</td>
</tr>
</tbody>
</table>

Table 6.4. Effect size (Cohen’s d) for each sample and for Q1 and Q2. In general, there was a small effect for Accuracy Changed (Q1) and almost no effect for Accuracy Common (Q2).

<table>
<thead>
<tr>
<th>Sample</th>
<th>Accuracy Changed (Q1)</th>
<th>Accuracy Common (Q2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cohen’s d</td>
<td>Effect Size</td>
</tr>
<tr>
<td>1</td>
<td>0.487</td>
<td>Small</td>
</tr>
<tr>
<td>2</td>
<td>0.583</td>
<td>Medium</td>
</tr>
<tr>
<td>3</td>
<td>0.168</td>
<td>None</td>
</tr>
<tr>
<td>4</td>
<td>0.354</td>
<td>Small</td>
</tr>
<tr>
<td>5</td>
<td>0.432</td>
<td>Small</td>
</tr>
<tr>
<td>6</td>
<td>0.485</td>
<td>Small</td>
</tr>
<tr>
<td>7</td>
<td>0.070</td>
<td>None</td>
</tr>
<tr>
<td>8</td>
<td>0.338</td>
<td>Small</td>
</tr>
<tr>
<td>9</td>
<td>0.301</td>
<td>Small</td>
</tr>
<tr>
<td>10</td>
<td>0.195</td>
<td>None</td>
</tr>
<tr>
<td>11</td>
<td>0.253</td>
<td>Small</td>
</tr>
<tr>
<td>12</td>
<td>0.531</td>
<td>Medium</td>
</tr>
<tr>
<td>Exit Questionnaire</td>
<td>0.702</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Table 6.5. Effect size (Cohen’s d) for each sample and for Q3 and Q4. In general, there was a medium to large effect on Accuracy (Q3) and Understandability (Q4).

<table>
<thead>
<tr>
<th>Sample</th>
<th>Accuracy (Q3)</th>
<th>Understandability (Q4)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cohen’s d</td>
<td>Effect Size</td>
</tr>
<tr>
<td>1</td>
<td>-0.627</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>-0.594</td>
<td>Medium</td>
</tr>
<tr>
<td>3</td>
<td>-0.722</td>
<td>Medium</td>
</tr>
<tr>
<td>4</td>
<td>-0.720</td>
<td>Medium</td>
</tr>
<tr>
<td>5</td>
<td>-0.779</td>
<td>Medium</td>
</tr>
<tr>
<td>6</td>
<td>-0.656</td>
<td>Medium</td>
</tr>
<tr>
<td>7</td>
<td>-0.632</td>
<td>Medium</td>
</tr>
<tr>
<td>8</td>
<td>-0.669</td>
<td>Medium</td>
</tr>
<tr>
<td>9</td>
<td>-0.970</td>
<td>Large</td>
</tr>
<tr>
<td>10</td>
<td>-0.519</td>
<td>Medium</td>
</tr>
<tr>
<td>11</td>
<td>-0.888</td>
<td>Large</td>
</tr>
<tr>
<td>12</td>
<td>-0.610</td>
<td>Medium</td>
</tr>
<tr>
<td>Exit Questionnaire</td>
<td>-0.881</td>
<td>Large</td>
</tr>
</tbody>
</table>

For H1, Q1 has statistical significance in all but three cases allowing us to reject H1<sub>0</sub> in favor of H1<sub>a</sub> in most cases. srcDiff also has a smaller mean for Q1, indicating that srcDiff has less code incorrectly marked as changed. The effect size is small to medium. This allows us to conclude that in general, srcDiff view is perceived to produce source code with less common code marked as changed.

Only one null hypothesis can be rejected of H2. Therefore, no statistical inference can be made. However, Q3 is statistically significant in all cases allowing us to reject H3<sub>0</sub> (no difference in how excellent a job marking differences) in favor of H3<sub>a</sub> (a difference in how excellent a job marking differences). srcDiff view also possesses a greater mean which indicates that the srcDiff view is perceived to do a better job of capturing changes. Effect size is medium to large. Thus, we can conclude that the srcDiff view does a more excellent job at marking source code changes.
Results for H1-H3 provide support for Hypothesis 1. That is, srcDiff produces source-code changes with less common code marked as changed (H1), does not mark more changed as common than GNU diff (H2) and does a better job (H3). So, we conclude that the srcDiff view is perceived to produce a more accurate view of source code changes than GNU diff.

From the results of Q4, we can safely reject all H4_0 in favor of H4_a with srcDiff_\mu > diff_\mu. The effect size is medium to large. From this we can conclude that srcDiff is perceived to provide an easier to understand view of source code changes. This directly provides support for Hypothesis 2 and we can therefore claim that the srcDiff view is perceived to provide an easier to understand view of source code changes than GNU diff.

The results for Q5 are shown in the stack bar chart in Figure 6.1. Blue shows the percentage each were selected the most preferable, green the second most preferable, and red the least preferable. The results show that on average over all the samples and exit questionnaire srcDiff was overwhelming chosen (67%) as the first choice and very infrequently chosen for either the second (16%) and last choice (17%). The original and modified was the second most frequently chosen first choice (18%), however, it was also the largest last choice (59%). Although GNU diff was the least popular first choice (14%), it was by far the most popular second choice (62%). As the results provide positive support for srcDiff in Hypothesis 3, we are able to conclude that the srcDiff view presents a view of source code changes that is more preferable than GNU diff.
In conclusion, survey results provide positive results for srcDiff in all three hypotheses. That is, srcDiff view of source code is perceived as more accurate and easier to understand than GNU diff and it is more preferable than GNU diff.

6.3 Threats to Validity

The following threats to validity have been identified. As a survey was used, only twelve samples were chosen so that the survey could be conducted in a manageable amount of time necessary to obtain responses and quality answers [Deutskens, de Ruyter, Wetzels, Oosterveld 2004] [Galesic, Bosnjak 2009]. That is, adding questions would have made the survey too long and would have drastically lowered our response rate. While we could not cover all possible situations we selected changes to be representative as possible. Some participants completed the survey faster than possible (i.e., speeders). Adjusting for
speeders only benefited the statistical inference testing toward srcDiff slightly, so all participants were left in the study.

One threat to validity may be the experience of the participants. Only undergraduate and graduate students participated in the survey. Industry professionals may give different results. This threat is mitigated as some participants did have expert level programming experience, and many of the students have worked or are working in industry.

The rules are derived from examining real-world software projects and our experience as developers. Several situations could exist for which no current rule applies. We designate this for future work. However, for the rules presented in this paper, we are confident in their validity as shown by the evaluation.

We examined changes to KOffice in developing our rules. This could be a threat to over-fitting as the survey questions are from KOffice. However, our experience with changes in KOffice and other systems was used to develop the rules, while an in-depth analysis of KOffice was used to refine, add to, and evaluate them. For instance, GCC was used in the early development of the Nesting rules. As such the rules represent changes that typically occur across systems. The samples for the survey where constructed from a pool using a taxonomy, mining for the most common syntactic changes along with manual inspection and then they were chosen to be representative of the heuristics. One of the samples (9) used in the study involves a heuristic that we specifically derived from examination of KOffice. We did not see this type of change in other systems (but did not do an extensive search for other examples). Results for the t-test on this are no better than
the remaining questions and removing affects on **Q5** provides small changes (>0.6%) to the reported percentages in favor of srcDiff.

Comparison to a different tool may yield different results. The full evaluation compares srcDiff to additional tools.

Lastly, changes were gathered for only bug-fix revisions. All though these might not be representative of all changes, we believe that these are among the more complicated/non-trivial changes and therefore more than adequate.
CHAPTER 7

Evaluation

After running the preliminary study, a second study was designed (building upon the experience gained in the first) to address possible threats to validity with the first study. The following is a summary of what is changed.

- Comparison against multiple types of differencing tools including a syntactical differencer and textual differencers supporting within-line change highlighting.

- Questions were reworded to be less ambiguous and to accommodate the expanded study.

- Questions for accuracy, preference, and understandability were changed to constant-sum questions to allow for ranking of approaches in each criteria and to ascertain the degree of which they out rank each other.

- Comparison is in Java as the syntactical differencer does not support C++.

- Changes were collected from multiple projects uninvolved with rule creation.

- Changes were collected partially randomly from all change history with non-randomness used to try and diversify the selected changes and increase generalizability of results.

We now discuss the evaluation in depth.
In this study, srcDiff and its rules are compared to GumTree [Falleri et al. 2014], GitHub’s difference view [GitHub 2016] and Mergely [Peabody 2016].

For this study, we formulate three research questions.

• **RQ1:** Does srcDiff produce changes that are easier to read and understand than the other approaches?

• **RQ2:** Is srcDiff preferred over the other approaches?

• **RQ3:** Does srcDiff result in a delta that is more accurate than the other approaches?

GumTree is a syntactic-differencing approach. As computing an optimal edit script while considering moves is known to be NP-hard, GumTree uses heuristics to approximate the optimal-edit script. For details of the algorithm please see [Falleri et al. 2014]. Like most syntactic-differencing approaches, GumTree is unable to provide changes to comments and whitespace. GumTree can output a side-by-side view of changes by overlying the AST edits onto the source code. Several other non-visual representations of the edits are also provided. In [Falleri et al. 2014], GumTree is compared to several other differencers. The results show that GumTree produces a more optimal-edit script. In addition, the authors performed a manual investigation to compare the visual output of GumTree to that of Mergely. Once again, the results show GumTree as the best approach compared. As GumTree represent the best research tool currently available, GumTree is selected for this study.

GitHub’s difference view is a line-based differencing approach to source-code differencing that also highlights within-line changes. It supports both a side-by-side view
and a unified view of changes. It is limited in the fact that it can only be used to view changes in repositories hosted by GitHub. However, as it is used by GitHub, it sees heavy use and is therefore selected for study.

Mergely is another line-based differencing approach that supports highlighting of within-line changes. The within-line changes are computed in a separate manner than GitHub’s difference view. It supports a side-by-side view of changes. As Mergely was used in a comparison with GumTree [Falleri et al. 2014], it provides an excellent baseline and point of comparison between these studies.

As previous approaches aim at an optimal edit script, edit script size and number-of-mappings are often used to evaluate different syntactic-differencing approaches [Falleri et al. 2014; Hashimoto, Mori 2008]. However, such an evaluation is not appropriate here. Edit script size does not take into account the understandability of the resultant delta as we have demonstrated previously. Additionally, edit script cannot be used for comparing syntactic approaches to line-differencing approaches. Lastly, the AST used in srcDiff and GumTree differ drastically (e.g., different number of nodes, different syntactic categories) and a comparison of edit script is therefore not very meaningful.

Since the goal of srcDiff is to produce a delta that is both accurate and understandable to programmers, an online within-participant survey is conducted as an evaluation. The study consists only of side-by-side views of source-code changes in order to provide for a fair comparison with GumTree, which only supports a side-by-side view. srcDiff and GitHub also support a unified view. Replication package and research data for the study can be found at sdml.cs.kent.edu/srcDiff/dissertation/evaluation.
7.1 Data Collection and Preparation

Since, GumTree was evaluated on the Java [Falleri et al. 2014] programming language, Java is used as the language for samples in the study. In total, fourteen samples of source-code changes are collected from four separate Java systems: Elasticsearch, Google Guava, SLF4J, and jEdit. The following explains how change samples were collected for the study. In general, the collection procedure aims at getting a wider variety of changes to increase the generalizability of the results.

First, 100 revisions are selected at random for each system from all revisions with modified files. From these, two shorter methods, 5 to 30 lines, with smaller amounts of change, 2 to 10 lines as reported by a whitespace-ignoring line diff, and one larger method, 20 to 50 lines, with larger amounts of changes, 10 to 25 lines, are selected at random from Elasticsearch, Google Guava, and SLF4J. These criteria are used to provide for a wide coverage of change complexity while keeping the size of the methods and amount of change manageable for a survey. One sample was redrawn, as the change was too simple to produce variation in the output of each tool. The remaining five samples were chosen to supplement the randomly-drawn methods with different change types and selected by querying the 100 random revisions (via srcML on srcDiff documents). They are: a replaced method (Google Guava), a heavily modified class (Google Guava), and three examples of changes to conditions and conditionals (jEdit and Google Guava). All the samples can be found in APPENDIX D or as part of the replication package.

Ideally, a larger number of samples would have been included in the study, however, any larger of a size is prohibitive and greatly decrease the number of participants.
and quality of the results [Deutskens, de Ruyter, Wetzels, Oosterveld 2004; Galesic, Bosnjak 2009]. As such, jEdit is not used for the shorter and longer methods, as this generates too many examples for a survey. jEdit is used as part of the supplemental changes to provide both an additional system and a system using Subversion version control system (others utilized Git).

In the survey, GumTree is presented in HTML to allow for the interaction it supports via Javascript. To be comparable, srcDiff is also presented in HTML, although no Javascript interaction is needed. For GitHub and Mergely, we attempted to present them in HTML, however they are best presented as images. To control for any possible problems, directions are given in the survey to compensate for participants with small screens/resolutions. This worked well, as only three participants noted having trouble viewing GitHub/Mergely. One could not view them at all, so was removed from the pool of results, and the other two reported only inconvenience. For two with inconvenience, the results were analyzed with and without them. Since, removal of these two participants negatively affected GitHub and Mergely (clearly showing that having GitHub and Mergely as images was not a threat to validity) they were kept in the study. Several user responses about how much they liked GitHub’s view also shows that this was acceptable.

GumTree is unable to report changes to comments, so, comments are stripped from the samples. All the tools are run to ignore whitespace when computing a difference. Additionally, since GumTree is unable to report changes to whitespace, srcDiff is set to not report changes to whitespace. Unfortunately, GitHub and Mergely always show changes to whitespace and have no option to disable output of whitespace changes. As
such, participants are instructed to ignore changes to whitespace when answering questions. With exception of GumTree, all the tools are run on only the code samples. As GumTree requires syntactically-complete code, the code is wrapped with a class before it can be run. This is stripped out in the survey so that all tools show the same code. GumTree is also run on the full classes to check the results. There is no difference on ten of the fourteen samples. However, on the four remaining complete files, GumTree produced a slightly worse (sample 3, 5, 10) to far worse (sample 4) delta. On sample 13, GumTree reported two different changes depending on the run. The best is chosen. The latest release of GumTree (2.0.0) is used. The previous release (1.0.0) did not perform as well. That is, GumTree is shown in the best possible light for the evaluation.

Since, each tool outputs the differences with a slightly different representation (e.g., highlighting colors), the participants are given instructions on how the representations show the same concepts. Each sample is presented separately and the participant is given the output of the four approaches along with the original and modified code. Participants are instructed to answer in any order and only for the current sample. The samples are shown in random order. The study is blind with regard to the tools. That is, the approaches output is explained to the participants, however, the names of the approaches are not given.

7.2 Measures

The survey evaluated the approaches along four separate criteria: understandability (RQ1), readability (RQ1), personal preference (RQ2), and accuracy (RQ3). Table 7.1 contains a listing of the questions. The first two questions for accuracy use a standard unipolar rating scale of None at all, A slight amount, A moderate amount, A large amount,
and *A very large amount* (coded as 6-10). The remaining are constant sum questions, selected to allow for ties and to allow the participants to provide the extent for which a single approach is better than the other. All data is ordinal scale, that is, the median is calculated, while the mean is not valid [Good, Hardin 2012]. Lastly, the participants were given the option to provide comments. The questions are asked individually for each sample and after answering the questions for all samples, the participants are asked the same questions about the approaches overall.

**Table 7.1. Evaluation survey questions.**

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy - Unchanged</strong></td>
<td>How much of the code is marked as unchanged that should be marked otherwise?</td>
</tr>
<tr>
<td><strong>Accuracy - Changed</strong></td>
<td>How much of the code is marked as changed that should be marked otherwise?</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>Rank the tools by distributing 100 points among the tools relative to how well each tool does at capturing the differences between the original and modified code.</td>
</tr>
<tr>
<td><strong>Understandability</strong></td>
<td>Rank the tools by distributing 100 points among the tools relative to how easy each tool's output is to understand.</td>
</tr>
<tr>
<td><strong>Readability</strong></td>
<td>Rank the tools by distributing 100 points among the tools relative to how readable each tool's output is.</td>
</tr>
<tr>
<td><strong>Preference</strong></td>
<td>Rank the following by distributing 100 points among each relative to your preference.</td>
</tr>
</tbody>
</table>

**7.3 Results and Discussion**

The survey was given as a project in two software engineering classes and to a number of graduate students. Sixty-nine students (5 were experts) finished the survey. Table 7.2 and Table 7.3 show the results. As there is almost no variation in the first question, it is not reported. In preference, the participants are also asked to rank the
Original and Modified source code. As this received a median of zero in all cases, this is left out. All the research data and full charts are available in the replication package.

For understandability/readability questions, srcDiff is rated the best approach, most frequently (most points including ties), while GitHub is ranked second. This provides direct support for RQ1, that is, srcDiff is easier to read and understand. For the preference question, srcDiff is also rated best, most frequently (by a large margin). This also provides direct support for RQ2: srcDiff is preferred over the other approaches. In terms of accuracy, GumTree and srcDiff are tied. The reason is that the survey instructs participants to compare the approaches to what they determined is actually changed by the developer. Without enough experience examining changes to software, abstracting out what actually changed is a difficult task. That is, a large portion of the participants do not have significant experience with differencing tools and looking at changes relative to what a developer actually performed. While these participants are fully capable of evaluating the readability, understandability, and preference criteria, they did not have enough expertise to evaluate the accuracy. Results on experts (those rated as excellent skills using differencing tools) for the ranking questions, rate srcDiff as the most accurate 15/15 times, most understandable 15/15 times, most readable 15/15 times, and most preferred 14/15 times. No other tool is ranked the best more than 2 times on the combined samples and overall. In essence, GumTree appears more accurate to less skilled developers, possibly due to the additional fine-grained markup on different code, however, since these changes are not meaningful, they are unable to understand the changes. This supports RQ3. srcDiff
produces a delta that is more accurate than the other approaches. The authors invite the reader to view the changes produced by the approaches in the provided replication package.

Table 7.2. Results for each of the samples and post-questionnaire (Overall). All reported measures are medians with GitHub (GH), Mergely (M), srcDiff (SD), and GumTree (GT). Last line shows a count of the number of samples (and Overall) for which the approaches achieved the best result (including ties) out of all the approaches. For Changed, smaller number is better, others larger is better.

<table>
<thead>
<tr>
<th>Samples</th>
<th>Changed</th>
<th>Accuracy</th>
<th>Understandability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GH</td>
<td>M</td>
<td>SD</td>
</tr>
<tr>
<td>Sample 1</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Sample 2</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Sample 3</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Sample 4</td>
<td>8</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Sample 5</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Sample 6</td>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Sample 7</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Sample 8</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Sample 9</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Sample 10</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Sample 11</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Sample 12</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Sample 13</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Sample 14</td>
<td>7</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Overall</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Count</td>
<td>8</td>
<td>6</td>
<td>14</td>
</tr>
</tbody>
</table>

In [Falleri et al. 2014], GumTree is compared to Mergely via a manual investigation by the authors. Our study also supports that GumTree is better than Mergely. However, for preference, understandability, and readability, GitHub is more often rated higher than GumTree. GumTree is often ranked lower than all other approaches when presented with more complicated changes (e.g., sample 3, 5, 6), while in the GumTree study, only simple changes are used (those that ChangeDistiller reported as a single change). In contrast, srcDiff is ranked on par with or as the highest. This validates a major claim of this
dissertation: optimal tree-differences produce poor results on more complicated changes, while srcDiff performs very well.

Table 7.3. Results for each of the samples and post-questionnaire (Overall). All reported measures are medians with GitHub (GH), Mergerly (M), srcDiff (SD), and GumTree (GT). Last line shows a count of the number of samples (and Overall) for which the approaches achieved the best result (including ties) out of all the approaches. Larger is better.

<table>
<thead>
<tr>
<th>Samples</th>
<th>Readability</th>
<th>Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GH M SD GT</td>
<td>GH M SD GT</td>
</tr>
<tr>
<td>Sample 1</td>
<td>25 25 30 20</td>
<td>25 20 25 20</td>
</tr>
<tr>
<td>Sample 2</td>
<td>25 20 26 25</td>
<td>25 20 25 25</td>
</tr>
<tr>
<td>Sample 3</td>
<td>25 24 30 15</td>
<td>25 20 25 15</td>
</tr>
<tr>
<td>Sample 4</td>
<td>25 20 25 30</td>
<td>20 15 28 30</td>
</tr>
<tr>
<td>Sample 5</td>
<td>27 25 30 16</td>
<td>25 24 30 15</td>
</tr>
<tr>
<td>Sample 6</td>
<td>30 20 30 15</td>
<td>25 20 30 15</td>
</tr>
<tr>
<td>Sample 7</td>
<td>25 21 27 25</td>
<td>20 20 25 22</td>
</tr>
<tr>
<td>Sample 8</td>
<td>25 20 30 25</td>
<td>20 20 25 25</td>
</tr>
<tr>
<td>Sample 9</td>
<td>25 20 30 25</td>
<td>22 20 25 25</td>
</tr>
<tr>
<td>Sample 10</td>
<td>28 20 26 24</td>
<td>25 20 30 20</td>
</tr>
<tr>
<td>Sample 11</td>
<td>25 20 28 25</td>
<td>25 20 25 25</td>
</tr>
<tr>
<td>Sample 12</td>
<td>25 20 30 22.5</td>
<td>20 30 20</td>
</tr>
<tr>
<td>Sample 13</td>
<td>25 20 30 20</td>
<td>25 20 30 20</td>
</tr>
<tr>
<td>Sample 14</td>
<td>25 25 25 20</td>
<td>20 22 20 20</td>
</tr>
<tr>
<td>Overall</td>
<td>25 20 30 20</td>
<td>25 20 25 20</td>
</tr>
<tr>
<td>Count</td>
<td>3 1 13 1</td>
<td>5 1 13 5</td>
</tr>
</tbody>
</table>

The data was investigated for integrity and comments read. Nothing was suspicious about the expert’s data. From reading the comments on all participants, two people claimed inability to see code on the overall questions (which had no code, so possibly answered randomly), two had small inconvenience in viewing GitHub and GumTree images, and two may have made up their minds in the middle of the exam and thus stopped answering additional samples wholeheartedly (i.e., given up). The affect each had individually and combined was verified on the data. Removal of those who may not have answered overall
correctly changes the results slightly in favor of srcDiff. Those with some inconvenience with GitHub and Mergely images have a slight negative affect of them when removed. Those who may have given up had mostly slight affects. The result on the conclusions (all participants) is that srcDiff is slightly less accurate than GumTree, but GumTree and GitHub are even less preferable and understandable. This makes GumTree slightly more preferable than GitHub and GitHub roughly on par with Mergely. Looking at the data when removing all of these participants slightly benefits srcDiff. As these involve some amount of speculation, and the effects are minor, as they do not affect the conclusion of srcDiff being the most accurate (which comes from expert data) and the other criteria on either data set, these participants were left in with the presented data.

Also, of note the completion time was investigated for all participants. Two non-experts had a somewhat quick, but not impossibly quick response time. The data was also investigated without these two participants. As the results were in favor of srcDiff and do not affect the conclusions, they are also left in the presented data.

7.4 Timing

To illustrate srcDiff’s efficiency, we ran timings on srcDiff and GumTree using changes between 1.509.4 and 1.532.2 of the continuous integration server Jenkins (i.e., same releases as in GumTree [Falleri et al. 2014]). However, we did not discard any files, running both tools on all modified files in revisions between releases. In total, git log reports 1309 modified Java files. In contrast to [Falleri et al. 2014] and to better reflect a user’s expectations, we time entire runs of each using the time command. For srcDiff, a unified diff is computed (i.e., typical user request). As GumTree’s human-readable output
is to a URL requiring human interaction, its JSON output (latest revision March 2016 develop build) is used. The minimum of three runs on each file is taken. srcDiff took a mean wall clock time of 0.1 seconds, while GumTree took 1.17 seconds. Although GumTree’s human readable output is setup to listen to changes (i.e., subsequent comparisons faster than reported here), it is significantly slower in initial startup and thus in computing a delta for analysis. All timings run on iMac running El Capitan with 2.7GHz i5/8GB RAM.

7.5 Threats to Validity

The following threats to validity are identified. As a survey is used, only fourteen samples are chosen so that the survey can be conducted in a manageable amount of time necessary to obtain responses and quality answers [Deutskens, de Ruyter, Wetzels, Oosterveld 2004; Galesic, Bosnjak 2009]. Adding questions makes the survey too long and most likely drastically lower the response rate. To mitigate potential problems with response rate/quality, subjects are informed that they can do the survey in multiple sittings and given proper incentive. While all possible change situations cannot be covered, nine of the fourteen are drawn at random, and thus, represent more typical change situations. The remaining are used to supplement the randomly drawn in order to make sure certain types of changes are covered, e.g., changes to conditions and conditionals which we regard as very important for differencing. That is, we can claim that srcDiff is better in both the typical cases and important cases selected for the survey.

One threat may be that a single representation of changes was not used for all four approaches. If all the formats were unified into the GumTree view, then as the GitHub and
Mergely have no equivalent markup or report moves and updates, the conversion of such would have been arbitrary (biased). If all were unified into one of the other simpler views, then the possible accuracy that GumTree is afforded by the additional markup would be lost (bias) and could make the output more difficult to read and understand (bias). That is, we feel that the explanation of the different approaches is the best course of action.

In addition, accuracy is asked in regard to how well the developers changes are recovered and since the differences in markup of each type is explained, it is largely agnostic to how it is stylistically marked up. Understandability is asked in relation to how easy it is to understand the output. By definition, to understand is “to perceive the meaning” [Dictionary.com 2017]. So, the question asks how easily they can perceive the meaning of the output. As the participants were instructed in how each tool presents changes (i.e., they are equally capable of understanding how each change type is stylistically marked up), this is largely what is marked up and as what operation. To many, understandability and readability are the same measure (at least one comment in the data attests to this), to others they are not. As the results are close, they were regarded as mostly the same. Preference is based on the individual and their weighting of the other criteria.

One aspect that could affect ability to evaluate the approaches is color blindness, which is controlled for in the study. The removal of all color-blind participants is negligible and does not affect the conclusions.

One threat to validity may be the experience of the participants. A broader set of industry professionals may give different results. This threat is mitigated as some
participants have expert-level programming/differencing experience, and/or have worked in industry.

The rules are derived from examining real-world software projects and our experience as developers. Situations may exist for which no current rule applies. We designate this future work and note that srcDiff is highly adaptable (i.e., new rules easily added) and not fixed on immutable algorithms. We are confident in the rules presented, as shown by the evaluation.

srcDiff is developed primarily to support C/C++. Minimal effort went into adding support for Java. Different systems are used for investigation (android libcore, lucene-solr, and hadoop) and multiple systems for evaluation. As such, there is no over-fitting present.

Comparison to other approaches may yield different results. However, since we compared multiple approaches (line and state-of-the-art syntactic) we are confident results generalize.

People commit to different version control systems (VCS) in different granularities, especially centralized/decentralized systems. Both Subversion and Git are used in the survey/investigation, although Subversion is used to a lesser extent.
CHAPTER 8

Investigating Method Stereotype History

In this chapter, the first application of srcDiff to a software engineering problem is presented. Broadly, the task is to investigate changes to method stereotypes. That is, we are interested in seeing how each individual method’s stereotype evolves over time. This study answers the following research questions:

- **RQ1:** Is a method’s stereotype stable?
- **RQ2:** What consequences are there to a method’s stereotype changing?

In Section 8.1, method stereotypes are introduced along with the relevant related work. In Section 8.2, the problem is redefined and stated in more detail. Section 8.3 provides the details of data collection. The results and discussion are given in Section 8.4 and threats to validity in Section 8.5. The general conclusions are provided in Section 8.6.

8.1 Method Stereotypes

A brief introduction to method stereotypes is provided in this section. For a more complete description see [Dragan, Collard, Maletic 2006]. The work of Dragan et al. [Dragan, Collard, Maletic 2006] introduces a taxonomy for method stereotypes. Table 8.1 shows a version of the taxonomy given by Dragan et al. [Dragan, Collard, Hammad, Maletic 2011; Dragan, Collard, Maletic 2010]. A method stereotype concisely represents behavioral aspects of a method. Method stereotypes are separated into five broad categories (far-left column):
- **Accessors** – query the state of an object on which it is called.
- **Mutators** – modify the state of an object on which it is called.
- **Creational** – create/provide new objects.
- **Collaborational** – work on objects pertaining to classes other than itself.
- **Degenerate** – structural/collaborational stereotypes are limited. Simplest stereotypes.

**Table 8.1. Method stereotype taxonomy.** The far-left column is the broad stereotype categories, the middle column the individual stereotypes, and the right column is a description of each stereotype

<table>
<thead>
<tr>
<th>Stereotype Category</th>
<th>Stereotype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structural Accessor</strong></td>
<td>get</td>
<td>Returns a data member.</td>
</tr>
<tr>
<td></td>
<td>predicate</td>
<td>Returns Boolean value that is not a data member.</td>
</tr>
<tr>
<td></td>
<td>property</td>
<td>Returns info about data members.</td>
</tr>
<tr>
<td></td>
<td>void-accessor</td>
<td>Returns information via a parameter.</td>
</tr>
<tr>
<td><strong>Structural Mutator</strong></td>
<td>set</td>
<td>Sets a data member.</td>
</tr>
<tr>
<td></td>
<td>command</td>
<td>Performs a complex change to the object’s state.</td>
</tr>
<tr>
<td></td>
<td>non-void-command</td>
<td></td>
</tr>
<tr>
<td><strong>Creational</strong></td>
<td>constructor, copy-const, destructor, factory</td>
<td>Creates and/or destroys objects.</td>
</tr>
<tr>
<td><strong>Collaborational</strong></td>
<td>collaborator</td>
<td>Works with objects (parameter, local or return value).</td>
</tr>
<tr>
<td><strong>Degenerate</strong></td>
<td>stateless</td>
<td>No data members read/written directly and can have only one call to other class methods</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>Has no statements.</td>
</tr>
</tbody>
</table>

The individual stereotypes (middle column with its description in the far-right column) indicate a refinement of the broad behavior described by the category. As an example, *accessor* methods (general category) query an object’s state. A predicate method
is a special type of accessor that returns a computed Boolean containing some information about the state of the object on which it is called.

Method stereotypes are not necessarily exclusive. That is, a method may have multiple different stereotypes. For example, it is common for any of the stereotypes to also have collaborator as a secondary stereotype. In addition, while considered an anti-pattern, a method can both query an object’s state (accessor) and modify that state as well (mutator).

In addition to the taxonomy, Dragan et al. [Dragan, Collard, Maletic 2006] also presents a set of rules for automatically identifying a method’s stereotype from its source-code. These rules are encoded and available in the publically available tool StereoCode (https://github.com/srcML/stereocode). StereoCode reports a superset of the stereotypes defined above. For instance, a pure get method is required to be const and thus cannot be both an accessor and mutator. However, the tool will also identify non-const get methods.

Method stereotypes have been used to support various software-analysis tasks. The distribution of the different types of method stereotypes forms the basis for class stereotypes [Dragan, Collard, Maletic 2010]. The stereotypes of added/deleted methods have been used to categorize commits [Dragan, Collard, Hammad, Maletic 2011]. In addition, class/method stereotypes have been used to formulate documentation for methods [Abid, Dragan, Collard, Maletic 2015] and classes [Moreno et al. 2013], and method and commit stereotypes have been used for commit-message generation [Cortes-Coy, Linares-Vasquez, Aponte, Poshyvanyk 2014].
The closest work to the work done here is Dragan et al. [Dragan, Collard, Hammad, Maletic 2011], which looked at only added and deleted methods to categorize commits, and Dragan [Dragan 2010] who looked at the distribution of method stereotypes in a few releases (~20) of two systems and found the distribution of method stereotypes to be fairly consistent for one system and unstable for the other. For this study, we will investigate the stability and consequences of the stereotypes to modified methods (methods that were changed in some way) using the entire history provided by version-control systems. That is, this study is a compliment and a large extension to Dragan’s work [Dragan 2010].

8.2 Evolution of a Method’s Stereotype

We now restate and expand upon the research problem and discuss theoretically the consequences of a change in method stereotype (RQ2). A manual investigation of the consequences is provided in Section 8.4.2.

Method stereotypes capture behavioral aspects of a method. That is, a method is created for a specific purpose/intent to accomplish a specific goal and a method stereotype is a high-level description of this purpose. Throughout the lifetime of a method, if it is well designed, a method’s purpose should not change, and thus a method’s stereotype should remain constant. That is, we hypothesize that method stereotypes are resilient to change (i.e., they are stable). Furthermore, certain types of changes in method stereotype can be either neutral, positive, or negative indicators. For example, a transition from non-\texttt{const get} to a \texttt{get} is an example of a positive transition, while the opposite, a transition from a \texttt{get} method to a non-\texttt{const get} method is a negative transition that indicates a degradation in the design of the system (i.e., poor design). Note, the addition of a non-\texttt{const get}
method is often necessary in programming and therefore valid. However, with the transition removing a `const` accessor (which is necessary) for a non-`const` `get` method is negative. In this research, we focus on changes in stereotype only. Table 8.2 and Table 8.3 contain a categorization of different types of important stereotype transitions. This categorization is not meant to be complete, but to highlight important transitions. Category-column contains the category name, description-column gives a description of what stereotype transitions are a part of that category, and notes-column gives notes about the significance along with if such transitions are generally positive, negative, or neutral. Group-column groups similar stereotype change categories. This provides a partial answer to RQ2. In Section 8.4.2, we will investigate this further in a manual investigation.

The goal of this study is then, to investigate how often stereotypes change (i.e., RQ1) and if they change, how often and what to do stereotypes change. To do this, the entire history of several systems are analyzed and data collected on changes (or lack-

<table>
<thead>
<tr>
<th>Group</th>
<th>Category</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unclassified</td>
<td>Move to/from Unclassified</td>
<td>Method transitioned to/from Unclassified</td>
<td>• Methods that cannot be classified lack a clear abstraction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• To unclassified (negative)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• From unclassified (positive)</td>
</tr>
<tr>
<td>Structural</td>
<td>Move to/from Non-const get</td>
<td>Method transitioned between non-const <code>get</code> and another <code>Accessor</code>.</td>
<td>• From <code>Accessor</code> to non-const <code>get</code> breaks ability use on constants/degrades design (negative).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• From non-const <code>get</code> to <code>Accessor</code> increases information hiding. Method most likely should have always been <code>Accessor</code> (positive).</td>
</tr>
</tbody>
</table>
### Table 8.3. Method stereotype transition categories (continued).

| Multiple Categories | Add/Remove/Replace Categories | Method that has multiple stereotypes from at least two of the following: *Structural Accessor*, *Structural Mutator*, and *Creatational*. | • Method has too much responsibility.  
• Presence is possible code smell.  
• Transition that adds additional method stereotypes indicates degrade in design.  
• Transition that removes stereotypes indicates a design improvement.  
  Transitions where categories are replaced with others indicates poor design. |
|---|---|---|---|
| Multiple Categories | Add/Remove/Replace Categories | Method that has multiple stereotypes from at least two of the following: *Structural Accessor*, *Structural Mutator*, and *Creatational*. | • Massive change to function behavior.  
• Change is suspicious and should be investigated. |
| Cross Stereotype Boundaries | Add/Remove/Replace Categories | Method’s that change from *Structural Accessor*, *Structural Mutator*, and *Creatational* to a different one of those categories. | • Massive change to function behavior.  
• Change is suspicious and should be investigated. |
| Degenerate | Remove Degenerate | A method removes a *Degenerate* stereotype or transitions from a *Degenerate* stereotype to one of another category. | • *Degenerate* methods do not provide enough functionality.  
• Removal is generally a positive.  
• Indicates addition of more functionality or increased complexity. |
| Collaborational | Remove Collaborational | A method removes a *Collaborational* stereotype or transitions from a *Collaborational* stereotype to one of another category. | • Removal of *Collaborational* indicates decreased dependency to other object(s). |
| Collaborational | Add Collaborational | A method adds a *Collaborational* stereotype or transitions from another category to a *Collaborational* stereotype. | • Addition of *Collaborational* indicates increased dependency to other object(s). |
| Degenerate | Add Degenerate | A method adds a *Degenerate* stereotype or transitions from another category to a *Degenerate* stereotype. | • Method’s functionality has been diminished.  
• Indicates method does not have enough responsibility and consider removal. |
| Collaborational | | | |
thereof) to method stereotypes. As the transition categories provided in Table 8.2/Table 8.3 is primarily theoretical, an investigation into the changes that induce a change in method stereotype will be investigated manually on a number of different examples.

8.3 Data Collection

To investigate changes in method stereotypes, fifty open-source software systems are selected for study. Table 8.4 and Table 8.5 provide a listing of the selected systems (left-column). The number of revisions in each that were non-merges and contained a modified C++ file are in the middle column, and the right column has the location from which the repository was taken. When the repository location is not a fully qualified URL, the repository comes from GitHub and has the following complete location: https://github.com/{Location}.git. The selected systems cover many different domains and repository sizes and therefore, provide a good representative sample.

The data collection process consists of the following parts:

1. Identification of all revisions and files changed in those revisions.
2. Application of srcDiff to the original and modified version of all changed C++ files to generate srcDiff archives of each revision.
3. Analysis of the srcDiff archives to locate changed methods.
4. Collecting of the original and modified stereotype of each changed method.

More specifically, data is collected for each system in the following manner. First, a local clone of the repository is taken. Next, a srcDiff archive of every modified C++ file
Table 8.4. List of selected software systems by alphabetical order. Left column is system, middle is number of revisions with modified C++ files, and right is the repository location. Unqualified locations are from GitHub: https://github.com/{Location}.git.

<table>
<thead>
<tr>
<th>System</th>
<th># Revisions</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>antimony</td>
<td>1,737</td>
<td>mkeeter/antimony</td>
</tr>
<tr>
<td>appleseed</td>
<td>5,369</td>
<td>appleseedhq/appleseed</td>
</tr>
<tr>
<td>BansheeEngine</td>
<td>3,058</td>
<td>BearishSun/BansheeEngine</td>
</tr>
<tr>
<td>bitcoin</td>
<td>5,785</td>
<td>bitcoin/bitcoin</td>
</tr>
<tr>
<td>blender</td>
<td>24,215</td>
<td>git://git.blender.org/blender.git</td>
</tr>
<tr>
<td>cgal</td>
<td>39,583</td>
<td>CGAL/cgal</td>
</tr>
<tr>
<td>ChaiScript</td>
<td>1,312</td>
<td>ChaiScript/ChaiScrip</td>
</tr>
<tr>
<td>citra</td>
<td>2,908</td>
<td>citra-emu/citra</td>
</tr>
<tr>
<td>Clang</td>
<td>56,631</td>
<td><a href="http://llvm.org/git/clang">http://llvm.org/git/clang</a></td>
</tr>
<tr>
<td>cocos2d-x</td>
<td>13,890</td>
<td>cocos2d/cocos2d-x</td>
</tr>
<tr>
<td>codeblocks</td>
<td>7,484</td>
<td>jenslody/codeblocks</td>
</tr>
<tr>
<td>code-lite</td>
<td>7,918</td>
<td>eranif/code-lite</td>
</tr>
<tr>
<td>CRYENGINE</td>
<td>1,092</td>
<td>CRYTEK/CRYENGINE</td>
</tr>
<tr>
<td>CTK</td>
<td>2,823</td>
<td>commontk/CTK</td>
</tr>
<tr>
<td>Dealii</td>
<td>20,340</td>
<td>dealii/dealii</td>
</tr>
<tr>
<td>distortos</td>
<td>2,498</td>
<td>DISTORTEC/distortos</td>
</tr>
<tr>
<td>Dlib</td>
<td>4,366</td>
<td>davisking/dlib</td>
</tr>
<tr>
<td>engine</td>
<td>2,078</td>
<td>flutter/engine</td>
</tr>
<tr>
<td>folly</td>
<td>3,277</td>
<td>facebook/folly</td>
</tr>
<tr>
<td>GamePlay</td>
<td>1,354</td>
<td>gameplay3d/GamePlay</td>
</tr>
<tr>
<td>gnuradio</td>
<td>3,191</td>
<td>gnuradio/gnuradio</td>
</tr>
<tr>
<td>griefly</td>
<td>1,074</td>
<td>griefly/griefly</td>
</tr>
<tr>
<td>irods</td>
<td>2,860</td>
<td>irods/irods</td>
</tr>
<tr>
<td>json</td>
<td>1,049</td>
<td>nlohmann/json</td>
</tr>
<tr>
<td>kdevelop</td>
<td>16,663</td>
<td>KDE/kdevelop</td>
</tr>
<tr>
<td>kokkos</td>
<td>745</td>
<td>kokkos/kokkos</td>
</tr>
<tr>
<td>Mantella</td>
<td>1,320</td>
<td>Mantella/Mantella</td>
</tr>
<tr>
<td>Mongo</td>
<td>26,345</td>
<td>mongodb/mongo</td>
</tr>
<tr>
<td>MultiMC5</td>
<td>1,267</td>
<td>MultiMC/MultiMC5</td>
</tr>
<tr>
<td>natron</td>
<td>10,064</td>
<td>MrKepzie/Natron</td>
</tr>
<tr>
<td>newton-dynamics</td>
<td>3,124</td>
<td>MADEAPPS/newton-dynamics</td>
</tr>
<tr>
<td>nix</td>
<td>2,704</td>
<td>NixOS/nix</td>
</tr>
<tr>
<td>ogre</td>
<td>5,280</td>
<td>OGREcave/ogre</td>
</tr>
<tr>
<td>oio</td>
<td>2,445</td>
<td>OpenImageIO/oio</td>
</tr>
<tr>
<td>ola</td>
<td>4,470</td>
<td>OpenLightingProject/ola</td>
</tr>
<tr>
<td>opencv</td>
<td>11,328</td>
<td>opencv/opencv</td>
</tr>
<tr>
<td>openFrameworks</td>
<td>6,620</td>
<td>openframeworks/openFrameworks</td>
</tr>
</tbody>
</table>
Table 8.5. Selected systems (continued).

<table>
<thead>
<tr>
<th>System</th>
<th># Revisions</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma</td>
<td>1,395</td>
<td>H-uru/Plasma</td>
</tr>
<tr>
<td>ppsspp</td>
<td>15,272</td>
<td>hrydgard/ppsspp</td>
</tr>
<tr>
<td>Qt</td>
<td>28,069</td>
<td>qt/qt</td>
</tr>
<tr>
<td>QuantLib</td>
<td>8,526</td>
<td>lballabio/QuantLib</td>
</tr>
<tr>
<td>Rcpp</td>
<td>1,947</td>
<td>ReppCore/Rcpp</td>
</tr>
<tr>
<td>rdkit</td>
<td>2,474</td>
<td>rdkit/rdkit</td>
</tr>
<tr>
<td>stellarium</td>
<td>8,677</td>
<td>Stellarium/stellarium</td>
</tr>
<tr>
<td>supertux</td>
<td>4,283</td>
<td>SuperTux/supertux</td>
</tr>
<tr>
<td>swift</td>
<td>30,343</td>
<td>apple/swift</td>
</tr>
<tr>
<td>tera</td>
<td>832</td>
<td>baidu/tera</td>
</tr>
<tr>
<td>tfs-old-svn</td>
<td>4,665</td>
<td>otland/tfs-old-svn</td>
</tr>
<tr>
<td>Urho3D</td>
<td>5,225</td>
<td>urho3d/Urho3D</td>
</tr>
<tr>
<td>xbmc</td>
<td>25,280</td>
<td>xbmc/xbmc</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>445,255</strong></td>
<td></td>
</tr>
</tbody>
</table>

for each revision of the default branch is computed (parts 1 and 2). This is done via a short Python program written specifically for this task. In order to generate the srcDiff archives for each revision, the Python program generates a list of each revision and the modified C++ files contained within each of those revisions (part 1). To generate this list, the Python program uses git-log. With the correct options, git-log will report (in a manner that can be processed by the Python program) all modified files and their associated revisions. Since we are interested in changes to existing methods and added and delete files can only contain added and deleted methods, added and deleted files are ignored. For similar
reasons, copied and renamed files are also ignored. In addition, `git-log` is set to not report merge revisions. As `git-log` reports modified files irrespective of contents, when processing the report provided by `git-log`, the Python program uses file extensions to identify C++ files and filter out non-C++ files. The file extensions explicitly supported by srcML for C++ are used with `.h` (a typical C-language extension that is still used in C++) treated as C++ code. Then, the Python program runs srcDiff (without any approximation) on all the identified modified C++ files for each revision and generates a single srcDiff archive for each revision (part 2).

Once the srcDiff archive is created for each revision, the data can be examined for stereotype changes (parts 3 and 4). To examine, the srcDiff files, a separate Python program was written to parse the srcDiff archive, identify and collect the signatures of changed methods (part 3), and then apply StereoCode to compute the original and modified method stereotypes for the changed method (part 4). In order to identify changed methods and collect their signatures, a Simple API for XML (SAX) [SAX 2001] parsing technique is used. A method is marked as changed and the signature collected if, it contains a change to the text (added, deleted, or modified) comprising the method, the text changed is not whitespace or part of a comment, and the method itself is not inserted or deleted.

---

6 A renamed file may also be modified. Git has heuristics to detect these, however, for this experiment only files explicitly reported by `git-log` as modified are used. Type changed, unmerged, had their pairings broken, and unknown status files are also ignored.
Whitespace and comments (i.e., non-source code) changes are ignored, as they are non-functional/stylistic changes. In addition to the signature, for each changed method, the line number, class the method belongs to if part of a class definition, and the file containing the method are also collected\(^7\). These additional attributes are collected to make a unique ID for each changed method.

After the information for each changed method is collected, \texttt{srcml} is used on the srcDiff archive to generate the srcML archive of the original and modified versions. Then, \texttt{StereoCode} is used on the original and modified srcML archives to compute the stereotype of each method. \texttt{StereoCode} can report these in a CSV format containing the method stereotype and the same ID information collected for the changed methods (line number, file name, class name, and method signature). This is used to lookup the original and modified stereotypes. The reason that \texttt{StereoCode} is applied to the entire file as opposed to just the original and modified version of each method, is to avoid error due to any dependency \texttt{StereoCode} may have on contextual information. For example, if a method is defined within a class definition, and \texttt{StereoCode} is only applied to the method, then \texttt{StereoCode} will fail to recognize that the method is a method because of

\(^7\) As they can vary between the original and modified versions of code, both the original and modified versions of each of the ID elements (signature, line number, class name, and file name) is gathered.
the lack of any identification that it is so (i.e., has a class namespace as part of the name or is in a class definition).

Finally, information is recorded in two ways. First, each instance (stereotypes and ID information), is recorded to disk, and secondly running totals on the counts of the number of times each stereotype transition occurred is updated.

All programs written for the data collection process were tested for and verified for accuracy.

8.4 Results and Discussion

Data was collected from all investigated systems. In Section 8.4.1, empirical results are presented and in Section 8.4.2, the manual investigation results are presented.

8.4.1 Empirical Results

The full data is too large to put in the dissertation format, so for the full data see http://sdml.cs.kent.edu/srcDiff/dissertation/srddiff_applications/. Between all systems, there was a total of 1,361,348 changed methods resulting in 2,007 unique stereotype transitions. The top ten stereotype transitions are presented in Table 8.6. The from-column is the original version stereotype, to-column is the modified version stereotype, count-column is the number of occurrences, and %-column is the percentage out of all changed methods. The percentage drops sharply from collaborator command to itself being 43.7% of all transitions, down to 0.9% for collaborator non_void_command nonconstget to itself. The majority (1,243) of the unique transitions occurred only 10 times or less. All the top ten transitions showed no change in stereotype (i.e., agnostic to change).
Table 8.6. Top ten most common stereotype transitions. The from-column is original version stereotype, to-column is the modified version stereotype, count is the number of times it occurred, and % is the percentage of occurrence out of the all transitions.

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>count</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>collaborator command</td>
<td>collaborator command</td>
<td>595,085</td>
<td>43.7%</td>
</tr>
<tr>
<td>collaborator non_void_command</td>
<td>collaborator non_void_command</td>
<td>164,289</td>
<td>12.1%</td>
</tr>
<tr>
<td>collaborator property</td>
<td>collaborator property</td>
<td>76,635</td>
<td>5.6%</td>
</tr>
<tr>
<td>command</td>
<td>command</td>
<td>65,473</td>
<td>4.8%</td>
</tr>
<tr>
<td>collaborator</td>
<td>collaborator</td>
<td>63,473</td>
<td>4.7%</td>
</tr>
<tr>
<td>collaborator voidaccessor</td>
<td>collaborator voidaccessor</td>
<td>49,312</td>
<td>3.6%</td>
</tr>
<tr>
<td>collaborator</td>
<td>collaborator</td>
<td>40,183</td>
<td>3.0%</td>
</tr>
<tr>
<td>collaborator</td>
<td>collaborator get</td>
<td>23,614</td>
<td>1.7%</td>
</tr>
<tr>
<td>collaborator predicate</td>
<td>collaborator predicate</td>
<td>21,425</td>
<td>1.6%</td>
</tr>
<tr>
<td>collaborator non_void_command</td>
<td>collaborator non_void_command</td>
<td>11,926</td>
<td>0.9%</td>
</tr>
</tbody>
</table>

The top-ten transitions where the stereotype changed are shown in Table 8.7. The from-column is the original version stereotype, to-column is modified version stereotype, count is the number of occurrences, and % column is the percentage of times it occurred out of all transitions. For the most part, the transitions are neutral or positive (e.g., collaborator stateless property to collaborator property). The two types of transitions that maybe negative transitions are collaborator command to collaborator as the method may no longer be contributing much and collaborator non-void-command to collaborator property which indicates a significant deviation in behavior.

Also of note, each of the most frequent stereotype transitions with changes in stereotype are rare. Collectively, all stereotype transitions with changes in stereotype from
the original to modified versions is about 9.4%. This means that 90.6% of method changes resulted in no change to stereotype. With this we can now answer RQ1, method stereotypes change infrequently due to method changes and thus, method stereotypes are very stable.

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>count</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>collaborator</td>
<td>collaborator command</td>
<td>7,633</td>
<td>0.6%</td>
</tr>
<tr>
<td>command</td>
<td>collaborator command</td>
<td>6,327</td>
<td>0.5%</td>
</tr>
<tr>
<td>collaborator command</td>
<td>collaborator</td>
<td>4,761</td>
<td>0.3%</td>
</tr>
<tr>
<td>collaborator command</td>
<td>command</td>
<td>4,358</td>
<td>0.3%</td>
</tr>
<tr>
<td>collaborator</td>
<td>collaborator non VOID command</td>
<td>2,407</td>
<td>0.2%</td>
</tr>
<tr>
<td>collaborator non VOID command</td>
<td>collaborator command</td>
<td>2,257</td>
<td>0.2%</td>
</tr>
<tr>
<td>collaborator set</td>
<td>collaborator command</td>
<td>1,907</td>
<td>0.1%</td>
</tr>
<tr>
<td>collaborator non VOID command</td>
<td>collaborator property</td>
<td>1,889</td>
<td>0.1%</td>
</tr>
<tr>
<td>collaborator get</td>
<td>collaborator property</td>
<td>1,815</td>
<td>0.1%</td>
</tr>
<tr>
<td>collaborator stateless property</td>
<td>collaborator property</td>
<td>1,658</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

We now turn to how stereotypes change when they change. Over, 71% of the time that a non-const get method changed, it changed to a get. In fact, it is almost as common for a non-const get method to change into a get method, as it is to remain a non-const get. The opposite (get to non-const get) is exceedingly rare (22 total instances over all projects). That is, get methods are mistakenly written as non-const get methods and then corrected at a later date.
When a method stereotype changed, 52.7% of the time it only inserted or deleted one or more stereotypes. A summarization of the top-10 deleted and inserted groups of stereotypes are in Table 8.8 and Table 8.9 respectively. For both, the left-column is the stereotypes inserted or deleted, the middle-column is the number of occurrences, and the right-column is the percentage out of all the times method stereotypes changed (not the same between the original and modified versions). Also, for both, the most common consisted only of inserting or deleting a single stereotype. The addition of another stereotype (29.1%) is more frequent then removal of a stereotype (23.6%). In either case, collaborator is the most commonly inserted and deleted stereotype. When just considering method stereotypes that purely inserted or deleted stereotypes, this points to increased collaboration between classes in a system over time (the remaining pure insert and delete data that is not shown here does not offset this trend).

Table 8.8. Top ten purely deleted stereotypes. Left-column is the stereotypes deleted, the middle-column is the number of times those stereotypes were the only change in method stereotype, and the right-column is the percent out of all method stereotypes that changed.

<table>
<thead>
<tr>
<th>stereotype</th>
<th>count</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>collaborator</td>
<td>8,496</td>
<td>6.7%</td>
</tr>
<tr>
<td>stateless</td>
<td>7,498</td>
<td>5.9%</td>
</tr>
<tr>
<td>command</td>
<td>4,859</td>
<td>3.8%</td>
</tr>
<tr>
<td>non_void_command</td>
<td>2,283</td>
<td>1.8%</td>
</tr>
<tr>
<td>nonconstget</td>
<td>1,891</td>
<td>1.5%</td>
</tr>
<tr>
<td>factory</td>
<td>1,479</td>
<td>1.2%</td>
</tr>
<tr>
<td>collaborational_command</td>
<td>1,280</td>
<td>1.0%</td>
</tr>
<tr>
<td>empty</td>
<td>356</td>
<td>0.3%</td>
</tr>
<tr>
<td>set</td>
<td>269</td>
<td>0.2%</td>
</tr>
<tr>
<td>pure_stateless</td>
<td>242</td>
<td>0.2%</td>
</tr>
</tbody>
</table>
Table 8.9. Top ten purely inserted stereotypes. Left-column is the stereotypes inserted, the middle-column is the number of times those stereotypes were the only change in method stereotype, and the right-column is the percent out of all method stereotypes that changed.

<table>
<thead>
<tr>
<th>stereotype</th>
<th>count</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>collaborator</td>
<td>12,532</td>
<td>9.8%</td>
</tr>
<tr>
<td>command</td>
<td>7,705</td>
<td>6.0%</td>
</tr>
<tr>
<td>stateless</td>
<td>5,263</td>
<td>4.1%</td>
</tr>
<tr>
<td>non_void_command</td>
<td>3,618</td>
<td>2.8%</td>
</tr>
<tr>
<td>nonconstget</td>
<td>1,423</td>
<td>1.1%</td>
</tr>
<tr>
<td>collaborational_property</td>
<td>1,126</td>
<td>0.9%</td>
</tr>
<tr>
<td>property</td>
<td>1,037</td>
<td>0.8%</td>
</tr>
<tr>
<td>factory</td>
<td>1,000</td>
<td>0.8%</td>
</tr>
<tr>
<td>collaborational_command</td>
<td>827</td>
<td>0.6%</td>
</tr>
<tr>
<td>predicate</td>
<td>415</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

The remaining times a stereotype changed was 19.5% completely changed from one stereotype to another, and 27.8% one or more stereotypes were replaced with one or more other stereotypes. Table 8.10 and Table 8.11 provide the top ten completely changed method stereotypes and replaced stereotypes respectively.

In Table 8.10, the from-column is the original version method stereotype, the to-column is the modified version method stereotype, the count-column is the number of occurrences, and %-column is the percentage out of all method stereotypes that changed. The most frequently completely changed stereotypes are set to command (within same category), unclassified to command (positive), and non-const get to get (positive). Out of all the top ten, the only possibly negative transitions are those that are no longer able to classified by StereoCode, which may indicate that the methods have become convoluted, lacking a clear abstraction.
Table 8.10. Top ten completely changed stereotypes. The from-column is the original version stereotype, the to-column is the modified version stereotype, count-column is the number of occurrences and %-column is the percentage of occurrence out of all method stereotypes that changed.

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>count</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>command</td>
<td>1,567</td>
<td>1.2%</td>
</tr>
<tr>
<td>unclassified</td>
<td>command</td>
<td>1,519</td>
<td>1.2%</td>
</tr>
<tr>
<td>nonconstget</td>
<td>get</td>
<td>1,245</td>
<td>1.0%</td>
</tr>
<tr>
<td>unclassified</td>
<td>collaborator</td>
<td>1,077</td>
<td>0.8%</td>
</tr>
<tr>
<td>command</td>
<td>unclassified</td>
<td>955</td>
<td>0.7%</td>
</tr>
<tr>
<td>collaborator</td>
<td>unclassified</td>
<td>890</td>
<td>0.7%</td>
</tr>
<tr>
<td>command</td>
<td>set</td>
<td>856</td>
<td>0.7%</td>
</tr>
<tr>
<td>empty</td>
<td>collaborator</td>
<td>531</td>
<td>0.4%</td>
</tr>
<tr>
<td>stateless</td>
<td>unclassified</td>
<td>506</td>
<td>0.4%</td>
</tr>
<tr>
<td>empty</td>
<td>command</td>
<td>428</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

Table 8.11. Top ten replaced stereotypes. The from-column is what was replaced, the to-column what it was replaced with, count-column is the number of occurrences, and %-column is the percentage of occurrence out of all method stereotypes that changed.

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>count</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>non_void_command</td>
<td>command</td>
<td>3,003</td>
<td>2.4%</td>
</tr>
<tr>
<td>non_void_command</td>
<td>property</td>
<td>2,500</td>
<td>2.0%</td>
</tr>
<tr>
<td>set</td>
<td>command</td>
<td>1,907</td>
<td>1.5%</td>
</tr>
<tr>
<td>get</td>
<td>property</td>
<td>1,861</td>
<td>1.5%</td>
</tr>
<tr>
<td>command</td>
<td>non_void_command</td>
<td>1,519</td>
<td>1.2%</td>
</tr>
<tr>
<td>nonconstget</td>
<td>get</td>
<td>1,344</td>
<td>1.1%</td>
</tr>
<tr>
<td>property</td>
<td>get</td>
<td>1,212</td>
<td>0.9%</td>
</tr>
<tr>
<td>command</td>
<td>voidaccessor</td>
<td>1,147</td>
<td>0.9%</td>
</tr>
<tr>
<td>command</td>
<td>set</td>
<td>1,051</td>
<td>0.8%</td>
</tr>
<tr>
<td>empty</td>
<td>command</td>
<td>895</td>
<td>0.7%</td>
</tr>
</tbody>
</table>
As for Table 8.11, the most common replacement of stereotypes are *non-void-command* with *command* (neutral) or *property* (which, is worth looking into, as it could be adding a missing *const* or something more serious). The remaining are largely replacements from within the same category. With the exceptions, *empty* to *command* is a positive transition, and *command void-accessor* crosses categories and may be worth investigation.

In essence, method’s do not change stereotypes very often. When they do change, most of the most frequent types of changes are of little concern. There are, however, a few among the top-ten that may be worth investigating. Due to their relative rarity, an automatic detection and reporting/alerting tool may be of use. The design of which, we leave for future work.

**8.4.2 Manual Investigation**

In Section 8.2, we presented a theory on the consequences in a method’s stereotype changing. In Section 8.4.1, we showed that method stereotypes are stable to method change with a method’s stereotype changed roughly 10% of the time. In addition, the most frequent changes in method stereotype are generally safe and unsuspicious.

In this section, we manually investigate method changes that caused changes to method stereotypes. This method investigation utilized a preliminary copy of the empirical results. The approach taken is as follows. First, the developer of method stereotypes was contacted and asked to make notes about the changes in method stereotypes that we encountered. Next, we looked at one revision for each of the changes in method stereotype noted by the method stereotype developer, and stopped when we reached ten or less
instances of a particular method stereotype change. Manual investigation is a slow and tedious process. In order to make the investigation process more manageable srcDiff’s unified difference view and StereoCode were used to locate the changed methods.

The results of the manual investigation are fifty-seven methods contained within thirty-five revisions from eight of the investigated repositories. Some revisions contained multiple versions of the same stereotype change. They were also mostly identical changes to the methods. In order that one group is not over-represented by the same change in method stereotype only one instance from each is used. One revision contained a srcDiff error where the wrong methods were paired together resulting in an incorrect report of a method-stereotype change. The incorrect srcDiff output was unrelated to the srcDiff rules and presented algorithm. The fault is fixed for the empirical results presented in the previous section. Another contained a srcML error. In total, there were 33 changes in method stereotypes investigated. Table 8.12 presents a summary of the results with each of the changes in method stereotype grouped according to the taxonomy presented in Section 8.2. Other consisted of insignificant changes not included in the taxonomy (four instances of changes within the same stereotype category and one command collaborator to collaborator). In addition, one method change included both the Cross Stereotype Boundary and Multiple Category Stereotype, and is counted as part of both. For each of the categories present, a count of the number of transitions belonging to those categories along with a count and percentage of the number of times those changed methods were considered of no concern (Benign) or worthy of inspection by a lead developer or manager (Suspicious). The Benign category were typically neutral changes or ones that fixed a bug
or code smell. **Suspicious** changes were changes that were clearly wrong or indicated a problem with the function itself. With corrections for the one change counted twice, 32% of the changes in method stereotype were **Suspicious**, while 68% were **Benign**.

In agreement with the taxonomy of Section 8.2, **Remove Degenerate, Add Collaborational, Remove Collaborational, Non-const get** (convert from non-const get to get), and Other were not negative indicators. The one instance of stereotype becoming **unclassified** was suspicious (method is poorly designed). **Cross Stereotype Boundaries**, and **Add/Remove/Replace Categories** (both those that become ones and those that lost the multiple categories) were mixed. That is, the method stereotype change belonging to the **Cross Stereotype Boundaries** and **Add/Remove/Replace Categories** are indicators of problems with the method design or change itself, but only a portion of the time.

**Table 8.12. Summary of manual investigation.** The category-column contains method stereotype change categories. Count is the number of times that category appeared. The suspicious-column contains the count and percent of the number of times that category had a change that should be investigated more deeply, and the benign-column is the count and percentage of the times that category was of no concern.

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
<th>Suspicious</th>
<th>Benign</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>To Unclassified</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>From Non-const get</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Add Collaborational</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Remove Collaborational</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Remove Degenerate</strong></td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>Add Categories</strong></td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>Remove Categories</strong></td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Cross Stereotype Boundaries</strong></td>
<td>11</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>33</td>
<td>11</td>
<td>23</td>
</tr>
</tbody>
</table>
From this, we can conclude that although not perfect, certain categories of method-stereotype changes (*To Unclassified*, *Cross Stereotype Boundaries*, and *Add/Remove/Replace Categories*) can be useful indicators for method design problems or inappropriate method changes. That is, aided by the fact that changes in method stereotype by themselves are rare with specific categories being even rarer, investigation by a developer would not require substantial time.

From the manual investigation, it is noted that a finer-grained categorization may produce more accurate predictions. For example, many of the *Cross Stereotype Boundaries* did so because the method was made `const` or the `const` was removed. The addition of `const` in these cases, allowed *StereoCode* to make a correct assessment of behavior and were positive changes, while the removal of `const` was more negative. That is, refining the *Cross Stereotype Boundaries* into cases where adding `const` or removing `const` will increase prediction ability. Similar types of improvements may be made to *Add/Remove/Replace Categories*.

With this we finish answering **RQ2**. That is, changes in method stereotypes are rare. When they do change, many categories of changes are of little concern. However, other categories of changes in method stereotypes (*To Unclassified*, *Cross Stereotype Boundaries*, and *Add/Remove/Replace Categories*) can, a portion of the time, indicate method design problems and inappropriate changes to code.

8.5 **Threats to Validity**

The following threats to validity are identified with the study.
First, srcDiff, srcML, Git, and other tooling used as part of the study are complicated pieces of software, and as such contain faults. The parsing of C++ code is notoriously difficult, in part, due to preprocessor statements making C++ a non-context free grammar. The extent to which the tools report inappropriate ASTs is not known, yet srcML has a rigorous test suite and srcDiff is also well tested. Despite this, a negligible number of revisions failed to report a correct XML document (possible due to program crash, etc.). A negligible number of commits will not affect the vast amount of data collected.

Only standard C++ extensions were used to locate C++ files. Most of the projects used only these extensions, however, some used unusual extensions which possibly contained C++ code. For example, a C++ extension followed by .in, which would possible denote a file that is used to generate code as part of the build system, and .tpp, which could possible contain template code. These were only noticed in a few of the repositories. For future work, these files can be examined more closely and a determination made as to whether they should be included as part of the data collected.

The .h extension was treated as C++, however, it is used both as a C and C++ header extension. Some of the projects included both C and C++ code and used .h for both. As C++ is largely a super-set of C++ (with a few exceptions), the parsing of C code as C++ code is a minor threat.

srcML provides support for the markup of Qt extensions. Currently, these are always on. Some code may have been incorrectly converted to srcML when a non-Qt
project used an identifier that is part of the Qt extensions. In the future, it is planned for srcML to support Qt extensions optionally.

Some of the projects have committed code that was auto-generated by some tooling. Auto-generated files may exist throughout the history of a project or may have appeared and disappeared at any point in the history of the software. For the most part, we are uninterested in changes to auto-generated code, however, as they can appear at any point in history, they may be difficult to find. Investigation into auto-detection of all generated files through the history of projects can be a valid research topic. With the systems used as part of the study, two instances of auto-generated files were found. One was in ppsspp, and the other in codelite. They were found due to srcDiff’s current difficulty in handling files with, for example, large static arrays and string literals, as they result in elements with exceedingly large numbers of children and text nodes for which Myers’ algorithm $O(ND)$ algorithm [Myers 1986] is not quick enough. In these two cases, the file list generation part of the data collection was instructed to ignore these auto-generated files.

Only C++ was used as part of the study. Results may be somewhat different for other languages. Currently, StereoCode only supports C++, and is thus a limitation.

Lastly, manual investigation is subject to human error. Mitigating this, the process was done carefully, and detailed notes taken. When the encoder was unsure about the change, another expert programmer was consulted. In addition, a breadth instead of depth look at the changes in method stereotypes is performed. One sample is certainly not representative of every possible change that could induce that particular change in
stereotype. That is, we can only conclude that changes in method stereotype can indicate problems, however, we cannot give the extent to which they do.

8.6 Conclusion

We investigated changes in method stereotype theoretically, and via a large empirical study and small manual investigation. A taxonomy of changes in method stereotypes was presented, and the manual investigation studied many different types of changes in stereotype belonging to those categories. We conclude that certain categories of method stereotype changes can be indicators of poor method design or of inappropriate changes to the methods themselves. In the empirical study, we conclude that changes in method stereotype are rare, and report various statistics on how and what changes when a method stereotype changes.
CHAPTER 9

Investigating Code Rewrite/Replacement and Code Conversion

This chapter presents the second application of srcDiff to a software engineering problem. Two aspects of programming (rewriting and replacing code and conversion of code from one element to another) are investigated. Both code replace and code conversion are explicitly supported by srcDiff and are easily identifiable in a srcDiff archive. As they are not supported directly (or at all) by other syntactical differencers, this study is a measure of how often srcDiff is able to capture information missed or marked incorrectly by other syntactical differencers.

The study answers the following research questions:

- **RQ1**: How commonly do developers replace and rewrite code?
- **RQ2**: What are the consequences of replace and rewriting code?
- **RQ3**: How often do developers convert code from one type to another?
- **RQ4**: What are the consequence of converting code?

The remainder of this chapter is laid out as following. Section 9.1 introduced code rewrite and replacement, how it is supported via srcDiff, and the significance. Section 9.2 explains code replacement, how srcDiff supports code conversion, and the significance. Section 9.3 explains the data collection procedures. Section 9.4 presented the results and discussion, the threats to validity are given in Section 9.5, and Section 9.6 presents the overall conclusions.
9.1 Rewrite and Replacement

When developing code, developers delete portions of code, expressions, statements, function, etc. and insert separate code in that location. This is what we term rewritten or replaced code. From now on, we will use rewrite and replace interchangeably.

Figure 9.1 shows an example code replacement with the original and modified code on the top, left and right respectively, and the srcDiff of the change on the bottom. This example is a simplified version of Martin Fowler’s example of a Substitute Algorithm refactoring [Fowler 1999]. The control-group (initialization, condition and increment) of the for-statement is the same between both versions, however, the body of the for-statement has been substituted with a different algorithm, i.e., it is replaced.

```
Original
for (int i = 0; i < people.length; i++) {
    if (people[i].equals("Don")){
        return "Don";
    }
    if (people[i].equals("John")){
        return "John";
    }
    if (people[i].equals("Kent")){
        return "Kent";
    }
}

Modified
List candidates = Arrays.asList(new String[] {"Don", "John", "Kent"});
for (int i = 0; i < people.length; i++)
    if (candidates.contains(people[i]))
        return people[i];
```

Figure 9.1. Example code replacement. The body of the for-statement is completely deleted and replaced with a different body.
The significance of this is that srcDiff is able to recognize the replacement, while a tree-edit distance based algorithm will achieve a shorter edit script by marking the body and if-statement components as modified in some manner. That is, by measuring how often replacement occurs, we get an additional measure of how much srcDiff outperforms tree-edit distance algorithms for syntactic differencing (RQ2).

Replacement can occur at any level of the Abstract Syntax Tree (AST). That is, at the file-level such as function replacement, at statement level, and at the expression/sub-expression level (e.g., identifier rename). In addition, AST elements can be replaced with separate AST elements, e.g., and if-statement can be replaced with a declaration-statement, and the number of elements deleted and inserted does not have to be equal. That is, a declaration-statement, if-statement, and while-statement, might be replaced with an expression-statement. The major restriction is that replaced elements are children of some common node (i.e., siblings) that occur in the same location.

srcDiff identifies these as a side-effect of the srcDiff rules. That is, when the rules fail and there are inserted and deleted children that are not identified as nested, converted, or previous versions of each other, they are marked as replacements. Figure 9.2 shows an example of how srcDiff reports code replacement. The delete portion of the replacement is marked with a diff:delete tag and type replace. Following after, the insert portion is reported with a diff:insert tag and type replace. In the example in Figure 9.2, an expression-statement, if-statement, and while-statement are replaced with a declaration-statement and a separate expression-statement. The contents of the statements are left out, and colors added for readability.
9.2 Conversion

When writing code, developers occasionally switch from using one syntactic element to another syntactic element of a separate type. As an example, a developer may have initially written a C++ struct with the intention of it only being used to store plain-old-data (similar to C usage). However, over time additional functionality such as constructors and methods are added to the struct such that it makes more sense to the developer to declare the same struct as a class instead. As another example, a statement original written as a for-statement, may be refactored at a later date to be a while-statement.

We call these code conversions. A code example is provided in Figure 9.3. In Figure 9.3, the original code is on the top-left, the modified code on the top-right, and the srcDiff is on the bottom. The original for-statement is converted into a while statement in the modified with the body of the for-statement left largely the same between both versions.
<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>int itemCount = d.items.count();</td>
<td>int itemCount = d.items.count() - 1;</td>
</tr>
<tr>
<td>for(int i = itemCount-1; i &gt;= 0; --i) {</td>
<td>while(i &gt;= 0) {</td>
</tr>
<tr>
<td>Item &amp;sbItem = d.items[i];</td>
<td>Item &amp;sbItem = d.items[i];</td>
</tr>
<tr>
<td>if (sbItem.widget() == widget) {</td>
<td>if (sbItem.widget() == widget) {</td>
</tr>
<tr>
<td>// several common lines</td>
<td>// several common lines</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>srcDiff</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>int itemCount = d.items.count() - 1;</td>
<td></td>
</tr>
<tr>
<td>for(i = itemCount-1; i &gt;= 0; i--) {</td>
<td></td>
</tr>
<tr>
<td>Item &amp;sbItem = d.items[i];</td>
<td></td>
</tr>
<tr>
<td>if (sbItem.widget() == widget) {</td>
<td></td>
</tr>
<tr>
<td>// several common lines</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9.3.** Example code conversion. The original code is on the top-left, modified top-right, and srcDiff on the bottom. In the example, the for-statement was converted into an equivalent while-statement. The body of the for-statement is largely unchanged.

srcDiff directly supports the detection and markup of code conversions. It does so for three specific reasons. First, for example, an if-statement and while statement are different AST elements, so a syntactic difference may never identify that the one element was converted to the other and identify the code as a replacement instead. Secondly, for example, not all instances of for-statements replaced with while-statements are conversions. That is, a tree-edit based syntactic difference could over-zealously pair parts of both elements and arrive at a difficult to understand delta. Lastly, without specific markup it can be difficult to detect that one element was converted to another, and to differentiate a convert with some other type of change operation (such as nesting). As an example, Figure 9.4 shows GumTree’s output on a simple code conversion [Falleri et al. 2014]. The example shows the original while-statement (top-left) converted into a for-statement (top-right). The bottom shows GumTree’s output. As GumTree does not support
code conversion, the while-statement is marked as deleted, for-statement inserted, and a series of moves between the two.

<table>
<thead>
<tr>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
</table>
| int i = 0;  
while(i < 10) {  
   System.out.println(i);  
   ++i;  
} | for(int i = 0; i < 10; ++i) {  
   System.out.println(i);  
} |

GumTree

![GumTree diagram](image)

**Figure 9.4.** Output of GumTree on a simple code conversion. Red is delete, yellow-green insert, and blue is move. A while-statement is converted to a for-statement. GumTree marks the while-statement deleted and for-statement inserted and a series of moves between the two.

Table 9.1 shows the conversions supported by srcDiff (repeated from Section 4.3.2). The left column is the category and the right column are the syntactic elements that are convertible with each other. foreach-statements are a Qt macro that were added to the Logical category beyond what was presented in Section 4.3.2.

Once detected, srcDiff outputs the conversion with a specific signature and markup. Figure 9.5 shows an example of a code conversion marked up in srcDiff. Colors are added and inner parts of the statements removed to simplify reading. The example is of a while-statement converted to a for-statement. In addition to the tags being marked with the type `convert`, a convert has a very specific pattern. It is always a `diff:delete` around the
original version element (while-statement) and keyword if present followed by a **diff:insert** around the modified version element (for-statement) and keyword if present. After this, the remaining parts then are marked with additional **diff:common**, **diff:insert**, and **diff:delete** tags as required.

Table 9.1. Code conversion supported by srcDiff. Left is category and right are the syntactic elements that can be converted to each other.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Syntactic Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Class</em></td>
<td>class, struct, union, enum</td>
</tr>
<tr>
<td><em>Access Regions</em></td>
<td>private, protected, public</td>
</tr>
<tr>
<td><em>Logical</em></td>
<td>if, while, for, foreach</td>
</tr>
<tr>
<td><em>Else</em></td>
<td>else, elseif</td>
</tr>
<tr>
<td><em>Statement</em></td>
<td>expr_stmt, decl_stmt, return</td>
</tr>
</tbody>
</table>

Figure 9.5. Example markup of a code conversion in srcDiff. Red is delete and green is insert. A **diff:delete** is wrapped around the original syntactic element (and keyword if present) with the type convert, and a **diff:insert** is wrapped around the modified syntactic element (and keyword if present) with type convert. The contents are then marked as common, inserted, and deleted as necessary.

9.3 Data Collection

This section is largely a repeat of Section 8.3. It is reproduced here to make the chapter more self-contained. The write-up deviates mostly after the discussion of how the srcDiff archives are created.
To investigate changes in code conversion and replacement, 50 open-source software systems are selected for study. Table 9.2 and Table 9.3 provide a listing of the selected systems (left-column). The number of revisions in each that were non-merges and contained a modified C++ file are in the middle column, and the right column has the location from which the repository was taken. When the repository location is not a fully qualified URL, the repository comes from GitHub and has the following complete location: https://github.com/{Location}.git. The selected systems cover many different domains and repository sizes and therefore, provide a good representative sample.

The data collection process consists of the following parts:

1. Identification of all revisions and files changed in those revisions.
2. Application of srcDiff to the original and modified version of all changed C++ files to generate srcDiff archives of each revision.
3. Analysis of the srcDiff archives to locate code replacement.
4. Analysis of the srcDiff archives to locate code conversion.

More specifically, data is collected for each system in the following manner. First, a local clone of the repository is taken. Next, a srcDiff archive of every modified C++ file for each revision of the default branch is computed (parts 1 and 2). This is done via a short Python program written specifically for this task. In order to generate the srcDiff archives for each revision, the Python program generates a list of each revision and the modified C++ files contained within each of those revisions (part 1). To generate this list, the Python program uses git-log. With the correct options, git-log will report (in a manner that can be processed by the Python program) all modified files and their associated revisions.
Table 9.2. List of selected software systems by alphabetical order. Left column is system, middle is number of revisions with modified C++ files, and right is the repository location. Unqualified locations are from GitHub: https://github.com/{Location}.git.

<table>
<thead>
<tr>
<th>System</th>
<th># Revisions</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>antimony</td>
<td>1,737</td>
<td>mkeeter/antimony</td>
</tr>
<tr>
<td>appleseed</td>
<td>5,369</td>
<td>appleseedhq/appleseed</td>
</tr>
<tr>
<td>BansheeEngine</td>
<td>3,058</td>
<td>BearishSun/BansheeEngine</td>
</tr>
<tr>
<td>bitcoin</td>
<td>5,785</td>
<td>bitcoin/bitcoin</td>
</tr>
<tr>
<td>blender</td>
<td>24,215</td>
<td>git://git.blender.org/blender.git</td>
</tr>
<tr>
<td>cgal</td>
<td>39,583</td>
<td>CGAL/cgal</td>
</tr>
<tr>
<td>ChaiScript</td>
<td>1,312</td>
<td>ChaiScript/ChaiScrip</td>
</tr>
<tr>
<td>citra</td>
<td>2,908</td>
<td>citra-emu/citra</td>
</tr>
<tr>
<td>Clang</td>
<td>56,631</td>
<td><a href="http://llvm.org/git/clang">http://llvm.org/git/clang</a></td>
</tr>
<tr>
<td>cocos2d-x</td>
<td>13,890</td>
<td>cocos2d/cocos2d-x</td>
</tr>
<tr>
<td>codeblocks</td>
<td>7,484</td>
<td>jenslody/codeblocks</td>
</tr>
<tr>
<td>codelite</td>
<td>7,918</td>
<td>eranif/codelite</td>
</tr>
<tr>
<td>CRYENGINE</td>
<td>1,092</td>
<td>CRYTEK/CRYENGINE</td>
</tr>
<tr>
<td>CTK</td>
<td>2,823</td>
<td>commontk/CTK</td>
</tr>
<tr>
<td>Dealii</td>
<td>20,340</td>
<td>dealii/dealii</td>
</tr>
<tr>
<td>distortos</td>
<td>2,498</td>
<td>DISTORTEC/distortos</td>
</tr>
<tr>
<td>Dlib</td>
<td>4,366</td>
<td>davisking/dlib</td>
</tr>
<tr>
<td>engine</td>
<td>2,078</td>
<td>flutter/engine</td>
</tr>
<tr>
<td>folly</td>
<td>3,277</td>
<td>facebook/folly</td>
</tr>
<tr>
<td>GamePlay</td>
<td>1,354</td>
<td>gameplay3d/GamePlay</td>
</tr>
<tr>
<td>gnuradio</td>
<td>3,191</td>
<td>gnuradio/gnuradio</td>
</tr>
<tr>
<td>griefly</td>
<td>1,074</td>
<td>griefly/griefly</td>
</tr>
<tr>
<td>irods</td>
<td>2,860</td>
<td>irods/irods</td>
</tr>
<tr>
<td>json</td>
<td>1,049</td>
<td>nlohmann/json</td>
</tr>
<tr>
<td>kdevelop</td>
<td>16,663</td>
<td>KDE/kdevelop</td>
</tr>
<tr>
<td>kokkos</td>
<td>745</td>
<td>kokkos/kokkos</td>
</tr>
<tr>
<td>Mantella</td>
<td>1,320</td>
<td>Mantella/Mantella</td>
</tr>
<tr>
<td>Mongo</td>
<td>26,345</td>
<td>mongodb/mongo</td>
</tr>
<tr>
<td>MultiMC5</td>
<td>1,267</td>
<td>MultiMC/MultiMC5</td>
</tr>
<tr>
<td>natron</td>
<td>10,064</td>
<td>MrKepzie/Natron</td>
</tr>
<tr>
<td>newton-dynamics</td>
<td>3,124</td>
<td>MADEAPPS/newton-dynamics</td>
</tr>
<tr>
<td>nix</td>
<td>2,704</td>
<td>NixOS/nix</td>
</tr>
<tr>
<td>ogre</td>
<td>5,280</td>
<td>OGRE/Cave/ogre</td>
</tr>
<tr>
<td>oio</td>
<td>2,445</td>
<td>OpenImageIO/oio</td>
</tr>
<tr>
<td>ola</td>
<td>4,470</td>
<td>OpenLightingProject/ola</td>
</tr>
<tr>
<td>opencv</td>
<td>11,328</td>
<td>opencv/opencv</td>
</tr>
<tr>
<td>openFrameworks</td>
<td>6,620</td>
<td>openFrameworks/openFrameworks</td>
</tr>
</tbody>
</table>
Table 9.3. Selected systems (continued).

<table>
<thead>
<tr>
<th>System</th>
<th># Revisions</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma</td>
<td>1,395</td>
<td>H-uru/Plasma</td>
</tr>
<tr>
<td>ppsspp</td>
<td>15,272</td>
<td>hrydgard/ppsspp</td>
</tr>
<tr>
<td>Qt</td>
<td>28,069</td>
<td>qt/qt</td>
</tr>
<tr>
<td>QuantLib</td>
<td>8,526</td>
<td>lballabio/QuantLib</td>
</tr>
<tr>
<td>Rcpp</td>
<td>1,947</td>
<td>RcppCore/Rcpp</td>
</tr>
<tr>
<td>rdkit</td>
<td>2,474</td>
<td>rdkit/rdkit</td>
</tr>
<tr>
<td>stellarium</td>
<td>8,677</td>
<td>Stellarium/stellarium</td>
</tr>
<tr>
<td>supertux</td>
<td>4,283</td>
<td>SuperTux/supertux</td>
</tr>
<tr>
<td>swift</td>
<td>30,343</td>
<td>apple/swift</td>
</tr>
<tr>
<td>tera</td>
<td>832</td>
<td>baidu/tera</td>
</tr>
<tr>
<td>tfs-old-svn</td>
<td>4,665</td>
<td>otland/tfs-old-svn</td>
</tr>
<tr>
<td>Urho3D</td>
<td>5,225</td>
<td>urho3d/Urho3D</td>
</tr>
<tr>
<td>xbmc</td>
<td>25,280</td>
<td>xbmc/xmbc</td>
</tr>
<tr>
<td>Total</td>
<td>445,255</td>
<td></td>
</tr>
</tbody>
</table>

Since we are interested in code conversion and code replace which only exist in modified files, added and deleted files are ignored. For similar reasons, copied and renamed files are also ignored.\(^8\) In addition, `git-log` is set to not report merge revisions. As `git-log` reports modified files irrespective of contents, when processing the report provided by `git-log`, the Python program uses file extensions to identify C++ files and filter out non-C++ files. The file extensions explicitly supported by srcML for C++ are used with `.h` (a typical C-language extension that is still used in C++) treated as C++ code. Then, the Python program runs srcDiff (without any approximation) on all the identified modified files.

\(^8\) A renamed file may also be modified. Git has heuristics to detect these, however, for this experiment only files explicitly reported by `git-log` as modified are used. Type changed, unmerged, had their pairings broken, and unknown status files are also ignored.
C++ files for each revision and generates a single srcDiff archive for each revision (part 2).

Once the srcDiff archive is created for each revision, the analysis of code replacement (part 3) and code conversion (part 4) is performed. A Python program is created for collecting code replacement and a separate Python program is created to collect code conversions. For code replacement, the Python program records a replace record each time a one diff:delete/diff::insert sequence with type replace is encountered. What is recorded, is the count of what elements were deleted, a count of what elements were inserted, as well as the original and modified parent elements. The parent element is gathered primarily to tell what kind of replacement occurred when text was replaced with text. For example, if text is replaced with text and both have a name tag as a parent, then a rename occurred. For a replace, comments appearing as part of the replace are counted, but whitespace is ignored. In addition, moves that occur as part of a replace are ignored. A replace is not counted unless a non-move and non-whitespace element was part of both the delete and insert parts. These records are then aggregated over all revisions while keeping track of the number of records and revisions. In addition, the number of times there was an element replaced with another one of the same type is recorded. When there are not the same number of elements (e.g., three expression statements inserted and two deleted), the minimum of these is counted (e.g., two). To further differentiate changes to text, changes to text of comments is marked as comment text, changes to text of a name as rename, changes to literal text as literal value, and everything else as other text. This information is then reported by the Python program in a series of
three CSV files (one for the deleted elements, one for the inserted elements, and a count of what elements were replaced with themselves) along with the number of revisions and replace records.

For convert analysis, each time a convert is encountered in the srcDiff archive, a count of the number of times the transition occurred is incremented. A map of maps is used to keep track of the transition counts. For example, a convert of a for-statement to while-statement is encountered in the srcDiff archive. Once encountered, the map is indexed first for the for-statement, which results in a second map. Then, the second map is index with the while-statement and the count is incremented. The convert counts for all revisions are then aggregated while keeping track of the number of revisions. This is then reported by the Python program in a CSV file.

All data analysis tools were tested to ensure accuracy.

9.4 Results and Discussion

Data was successfully gathered for all fifty systems. As each system resulted in a separate CSV file (one for each type of data collected), the data (of one type) for all systems were combined into a single table using JMP’s concatenate table feature. Apple’s Numbers program was then used to generate the data presented here. As the full tables are too large to fit into this dissertation format, they are available at http://sdml.cs.kent.edu/srcDiff/dissertation/srcdiff_applications/. The results and discussion for replace and convert are provided separately in Sections 9.4.1 and 9.4.2 respectively.
9.4.1 Replace Results and Discussion

Table 9.4 and Table 9.5 show the number of replaces for each system. The left column is the name of the system with the systems sorted alphabetically. The middle column is the number replaces (i.e., delete and insert pairs) that occurred, and the last column is the average number of replaces per revision (number of replaces divided by number of revisions). In total, there were over 4.8 million replaces across all revisions for all systems with an average of about 11 per revision. That per system average varied between system with distortos having the most per revision (61.5) and antimony having the least (3.0). The correlation between number of replaces and number of revisions is 0.88, and Figure 9.6 shows a plot of the number of revisions for a system on the x-axis with the number of replaces for that system on the y-axis. Both show that the number of replaces varies fairly linear with the number of revisions and is mostly likely fairly consistent.

With this information, we can partially answer RQ1. Code replacement as a whole is an extremely common event. This explains the poor results of tree-edit distance based syntactic differencers. The extent of this is, however, relative to the size of the structure that is replaced. That is, a rename is atomic, having only a text token replaced with another text token, and so, there is no error in a tree-edit distance approach, while a statement replace, function replace, or a replace of several different types with each other would show considerable deficiency by the syntactic difference.
Table 9.4. Number of replaces per system (alphabetically). First column is system, the second is the number of replaces, and the average per revision.

<table>
<thead>
<tr>
<th>System</th>
<th># Replaces</th>
<th># Per Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>antimony</td>
<td>5,242</td>
<td>3.0</td>
</tr>
<tr>
<td>appleseed</td>
<td>69,439</td>
<td>12.9</td>
</tr>
<tr>
<td>BansheeEngine</td>
<td>87,759</td>
<td>28.7</td>
</tr>
<tr>
<td>bitcoin</td>
<td>37,661</td>
<td>6.5</td>
</tr>
<tr>
<td>blender</td>
<td>200,596</td>
<td>8.3</td>
</tr>
<tr>
<td>cgal</td>
<td>491,979</td>
<td>12.4</td>
</tr>
<tr>
<td>ChaiScript</td>
<td>13,717</td>
<td>10.5</td>
</tr>
<tr>
<td>citra</td>
<td>21,357</td>
<td>7.3</td>
</tr>
<tr>
<td>Clang</td>
<td>357,243</td>
<td>6.3</td>
</tr>
<tr>
<td>cocos2d-x</td>
<td>261,194</td>
<td>18.8</td>
</tr>
<tr>
<td>codeblocks</td>
<td>87,576</td>
<td>11.7</td>
</tr>
<tr>
<td>codelite</td>
<td>107,021</td>
<td>13.5</td>
</tr>
<tr>
<td>CRYENGINE</td>
<td>27,786</td>
<td>25.4</td>
</tr>
<tr>
<td>CTK</td>
<td>22,806</td>
<td>8.1</td>
</tr>
<tr>
<td>Dealii</td>
<td>224,615</td>
<td>11.0</td>
</tr>
<tr>
<td>distortos</td>
<td>153,740</td>
<td>61.5</td>
</tr>
<tr>
<td>Dlib</td>
<td>23,783</td>
<td>5.4</td>
</tr>
<tr>
<td>engine</td>
<td>47,301</td>
<td>22.8</td>
</tr>
<tr>
<td>folly</td>
<td>19,864</td>
<td>6.1</td>
</tr>
<tr>
<td>GamePlay</td>
<td>32,845</td>
<td>24.3</td>
</tr>
<tr>
<td>gnuradio</td>
<td>28,620</td>
<td>9.0</td>
</tr>
<tr>
<td>griefly</td>
<td>6,845</td>
<td>6.4</td>
</tr>
<tr>
<td>irods</td>
<td>35,208</td>
<td>12.3</td>
</tr>
<tr>
<td>json</td>
<td>13,214</td>
<td>12.6</td>
</tr>
<tr>
<td>kdevelop</td>
<td>161,016</td>
<td>9.7</td>
</tr>
<tr>
<td>kokkos</td>
<td>8,809</td>
<td>11.8</td>
</tr>
<tr>
<td>Mantella</td>
<td>19,684</td>
<td>14.9</td>
</tr>
<tr>
<td>Mongo</td>
<td>223,300</td>
<td>8.5</td>
</tr>
<tr>
<td>MultiMC5</td>
<td>10,805</td>
<td>8.5</td>
</tr>
<tr>
<td>natron</td>
<td>194,015</td>
<td>19.3</td>
</tr>
<tr>
<td>newton-dynamics</td>
<td>174,221</td>
<td>55.8</td>
</tr>
<tr>
<td>nix</td>
<td>15,429</td>
<td>5.7</td>
</tr>
<tr>
<td>Ogre</td>
<td>78,849</td>
<td>14.9</td>
</tr>
<tr>
<td>oiiio</td>
<td>21,115</td>
<td>8.6</td>
</tr>
<tr>
<td>ola</td>
<td>49,425</td>
<td>11.1</td>
</tr>
<tr>
<td>opencv</td>
<td>134,100</td>
<td>11.8</td>
</tr>
<tr>
<td>openFrameworks</td>
<td>57,647</td>
<td>8.7</td>
</tr>
<tr>
<td>Plasma</td>
<td>66,014</td>
<td>47.3</td>
</tr>
<tr>
<td>ppsspp</td>
<td>166,901</td>
<td>10.9</td>
</tr>
<tr>
<td>Qt</td>
<td>295,292</td>
<td>10.5</td>
</tr>
</tbody>
</table>
Table 9.5. Number of replaces (continued).

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QuantLib</strong></td>
<td>106,641</td>
<td>12.5</td>
</tr>
<tr>
<td><strong>Rcpp</strong></td>
<td>39,345</td>
<td>20.2</td>
</tr>
<tr>
<td><strong>rdkit</strong></td>
<td>18,951</td>
<td>7.7</td>
</tr>
<tr>
<td><strong>stellarium</strong></td>
<td>80,393</td>
<td>9.3</td>
</tr>
<tr>
<td><strong>supertux</strong></td>
<td>37,320</td>
<td>8.7</td>
</tr>
<tr>
<td><strong>swift</strong></td>
<td>199,229</td>
<td>6.6</td>
</tr>
<tr>
<td><strong>tera</strong></td>
<td>10,655</td>
<td>12.8</td>
</tr>
<tr>
<td><strong>tfs-old-svn</strong></td>
<td>42,968</td>
<td>9.2</td>
</tr>
<tr>
<td><strong>Urho3D</strong></td>
<td>105,204</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>xbmc</strong></td>
<td>160,122</td>
<td>6.3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>4,854,861</td>
<td>10.9</td>
</tr>
</tbody>
</table>

Figure 9.6. The total number of revisions plotted against the number of replaces. X-axis is number of revisions and y-axis is the number of replaces. The result is fairly linear.
Next, the results of what elements are replaced with what elements are reported in Table 9.6, Table 9.7, and Table 9.8. For each of the three tables, the left column contains the element affected, the middle column the number of occurrences, and the last column is the number of occurrence divided by the number of revisions. Table 9.6 reports the elements that were deleted as part of a replace, Table 9.7 reports the elements inserted as part of a replace, and Table 9.8 reports the elements that were part of a replace with an element of the same type (self-replace). For each of the three tables, only the 20 most frequent are given. In addition, the normalized version of each table is also provided in Table 9.9, Table 9.10, and Table 9.11. For the normalized versions, the per revision for each system is computed and the median of all systems is presented. Normalization does not have a major affect on the data. That is, replace per revision is fairly consistent.

In all three cases, text is by far the most frequently replaced code element with over eleven text tokens deleted and inserted per revision (on average). In addition, renames occur about four times a revision (on average). That is, developers perform a lot of renames. Comment text is replaced fairly frequently (which most syntactic differencers do not report). After text, expressions and other subparts of statements are replaced with some relative frequency, followed by several types of statements and then functions.

Expression-statements and declaration-statements are the most frequently replaced statements. For expression-statements there are about 0.50 deleted and 0.42 inserted per revision (on average), which amounts to almost every other revision contained one as part of a replace. An expression-statement is replaced with itself (appeared in both the delete and insert) almost once every five revisions on average. Declaration-statements were
Table 9.6. Deleted elements in a replace. Left column is what element is deleted, the middle column is how many instances were deleted in all replaces, and the third column is the number each is replaced per revision.

<table>
<thead>
<tr>
<th>Element</th>
<th>Count</th>
<th>Per Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>4,992,841</td>
<td>11.21</td>
</tr>
<tr>
<td>name</td>
<td>428,616</td>
<td>0.96</td>
</tr>
<tr>
<td>expr</td>
<td>374,995</td>
<td>0.84</td>
</tr>
<tr>
<td>expr_stmt</td>
<td>221,339</td>
<td>0.50</td>
</tr>
<tr>
<td>operator</td>
<td>212,156</td>
<td>0.48</td>
</tr>
<tr>
<td>comment</td>
<td>209,506</td>
<td>0.47</td>
</tr>
<tr>
<td>call</td>
<td>166,444</td>
<td>0.37</td>
</tr>
<tr>
<td>decl_stmt</td>
<td>148,784</td>
<td>0.33</td>
</tr>
<tr>
<td>function</td>
<td>84,241</td>
<td>0.19</td>
</tr>
<tr>
<td>if</td>
<td>72,505</td>
<td>0.16</td>
</tr>
<tr>
<td>literal</td>
<td>65,426</td>
<td>0.15</td>
</tr>
<tr>
<td>parameter</td>
<td>64,408</td>
<td>0.14</td>
</tr>
<tr>
<td>cpp:value</td>
<td>60,002</td>
<td>0.13</td>
</tr>
<tr>
<td>function_decl</td>
<td>37,802</td>
<td>0.08</td>
</tr>
<tr>
<td>return</td>
<td>24,656</td>
<td>0.06</td>
</tr>
<tr>
<td>index</td>
<td>23,853</td>
<td>0.05</td>
</tr>
<tr>
<td>macro</td>
<td>21,808</td>
<td>0.05</td>
</tr>
<tr>
<td>argument_list</td>
<td>21,226</td>
<td>0.05</td>
</tr>
<tr>
<td>cpp:macro</td>
<td>20,881</td>
<td>0.05</td>
</tr>
<tr>
<td>cpp:define</td>
<td>19,906</td>
<td>0.04</td>
</tr>
</tbody>
</table>
Table 9.7. Inserted elements in a replace. Left column is what element is deleted, the middle column is how many instances were deleted in all replaces, and the third column is the number each is replaced per revision.

<table>
<thead>
<tr>
<th>Element</th>
<th>Count</th>
<th>Per Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>5,079,077</td>
<td>11.41</td>
</tr>
<tr>
<td>name</td>
<td>482,282</td>
<td>1.08</td>
</tr>
<tr>
<td>expr</td>
<td>379,892</td>
<td>0.85</td>
</tr>
<tr>
<td>comment</td>
<td>248,617</td>
<td>0.56</td>
</tr>
<tr>
<td>operator</td>
<td>216,847</td>
<td>0.49</td>
</tr>
<tr>
<td>expr_stmt</td>
<td>187,572</td>
<td>0.42</td>
</tr>
<tr>
<td>call</td>
<td>173,519</td>
<td>0.39</td>
</tr>
<tr>
<td>decl_stmt</td>
<td>154,274</td>
<td>0.35</td>
</tr>
<tr>
<td>if</td>
<td>66,061</td>
<td>0.15</td>
</tr>
<tr>
<td>function</td>
<td>64,533</td>
<td>0.14</td>
</tr>
<tr>
<td>cpp:value</td>
<td>60,009</td>
<td>0.13</td>
</tr>
<tr>
<td>parameter</td>
<td>59,126</td>
<td>0.13</td>
</tr>
<tr>
<td>function_decl</td>
<td>36,843</td>
<td>0.08</td>
</tr>
<tr>
<td>literal</td>
<td>34,263</td>
<td>0.08</td>
</tr>
<tr>
<td>argument_list</td>
<td>25,999</td>
<td>0.06</td>
</tr>
<tr>
<td>return</td>
<td>24,220</td>
<td>0.05</td>
</tr>
<tr>
<td>index</td>
<td>22,695</td>
<td>0.05</td>
</tr>
<tr>
<td>macro</td>
<td>22,283</td>
<td>0.05</td>
</tr>
<tr>
<td>cpp:macro</td>
<td>20,881</td>
<td>0.05</td>
</tr>
<tr>
<td>block</td>
<td>20,158</td>
<td>0.05</td>
</tr>
</tbody>
</table>
Table 9.8. Number times an element was replaced with itself. The left column is the element, the middle column is number of occurrence, and the right column is the number (on average) that occurred in each revision.

<table>
<thead>
<tr>
<th>Element</th>
<th>Count</th>
<th>Per Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>rename</td>
<td>1,780,920</td>
<td>4.00</td>
</tr>
<tr>
<td>comment text</td>
<td>1,278,269</td>
<td>2.87</td>
</tr>
<tr>
<td>other text</td>
<td>413,419</td>
<td>0.93</td>
</tr>
<tr>
<td>literal value</td>
<td>332,861</td>
<td>0.75</td>
</tr>
<tr>
<td>expr</td>
<td>328,508</td>
<td>0.74</td>
</tr>
<tr>
<td>name</td>
<td>156,090</td>
<td>0.35</td>
</tr>
<tr>
<td>call</td>
<td>88,444</td>
<td>0.20</td>
</tr>
<tr>
<td>expr stmt</td>
<td>79,809</td>
<td>0.18</td>
</tr>
<tr>
<td>cpp:value</td>
<td>59,909</td>
<td>0.13</td>
</tr>
<tr>
<td>operator</td>
<td>58,511</td>
<td>0.13</td>
</tr>
<tr>
<td>parameter</td>
<td>54,330</td>
<td>0.12</td>
</tr>
<tr>
<td>decl stmt</td>
<td>53,125</td>
<td>0.12</td>
</tr>
<tr>
<td>comment</td>
<td>34,739</td>
<td>0.08</td>
</tr>
<tr>
<td>if</td>
<td>26,762</td>
<td>0.06</td>
</tr>
<tr>
<td>cpp:macro</td>
<td>20,881</td>
<td>0.05</td>
</tr>
<tr>
<td>function</td>
<td>16,924</td>
<td>0.04</td>
</tr>
<tr>
<td>index</td>
<td>16,667</td>
<td>0.04</td>
</tr>
<tr>
<td>return</td>
<td>15,858</td>
<td>0.04</td>
</tr>
<tr>
<td>else</td>
<td>10,158</td>
<td>0.02</td>
</tr>
<tr>
<td>function decl</td>
<td>8,771</td>
<td>0.02</td>
</tr>
</tbody>
</table>
Table 9.9. Normalized deleted elements in a replace. The left-column is the deleted element. The right-column (normalized per revision) is calculated as: for each system, each element is normalized by the number of revisions for that system and the median across all systems is taken.

<table>
<thead>
<tr>
<th>Element</th>
<th>Normalized Per Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>10.65</td>
</tr>
<tr>
<td>name</td>
<td>0.84</td>
</tr>
<tr>
<td>expr_stmt</td>
<td>0.54</td>
</tr>
<tr>
<td>expr</td>
<td>0.44</td>
</tr>
<tr>
<td>operator</td>
<td>0.42</td>
</tr>
<tr>
<td>comment</td>
<td>0.41</td>
</tr>
<tr>
<td>call</td>
<td>0.36</td>
</tr>
<tr>
<td>decl_stmt</td>
<td>0.32</td>
</tr>
<tr>
<td>function</td>
<td>0.18</td>
</tr>
<tr>
<td>if</td>
<td>0.17</td>
</tr>
<tr>
<td>literal</td>
<td>0.14</td>
</tr>
<tr>
<td>parameter</td>
<td>0.12</td>
</tr>
<tr>
<td>function_decl</td>
<td>0.08</td>
</tr>
<tr>
<td>index</td>
<td>0.05</td>
</tr>
<tr>
<td>return</td>
<td>0.05</td>
</tr>
<tr>
<td>macro</td>
<td>0.04</td>
</tr>
<tr>
<td>for</td>
<td>0.04</td>
</tr>
<tr>
<td>argument_list</td>
<td>0.03</td>
</tr>
<tr>
<td>block</td>
<td>0.03</td>
</tr>
<tr>
<td>cpp:macro</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Table 9.10. Normalized inserted elements in a replace. The left-column is the inserted element. The right-column (normalized per revision) is calculated as: for each system, each element is normalized by the number of revisions for that system and the median across all systems is taken.

<table>
<thead>
<tr>
<th>Element</th>
<th>Normalized Per Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>10.41</td>
</tr>
<tr>
<td>name</td>
<td>1.00</td>
</tr>
<tr>
<td>expr_stmt</td>
<td>0.51</td>
</tr>
<tr>
<td>comment</td>
<td>0.46</td>
</tr>
<tr>
<td>operator</td>
<td>0.41</td>
</tr>
<tr>
<td>expr</td>
<td>0.39</td>
</tr>
<tr>
<td>call</td>
<td>0.36</td>
</tr>
<tr>
<td>decl_stmt</td>
<td>0.28</td>
</tr>
<tr>
<td>function</td>
<td>0.15</td>
</tr>
<tr>
<td>if</td>
<td>0.14</td>
</tr>
<tr>
<td>parameter</td>
<td>0.11</td>
</tr>
<tr>
<td>function_decl</td>
<td>0.09</td>
</tr>
<tr>
<td>literal</td>
<td>0.07</td>
</tr>
<tr>
<td>return</td>
<td>0.05</td>
</tr>
<tr>
<td>index</td>
<td>0.05</td>
</tr>
<tr>
<td>argument_list</td>
<td>0.04</td>
</tr>
<tr>
<td>macro</td>
<td>0.04</td>
</tr>
<tr>
<td>block</td>
<td>0.04</td>
</tr>
<tr>
<td>decl</td>
<td>0.03</td>
</tr>
<tr>
<td>for</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Table 9.11. Normalized times an element was replaced with itself. The left-column is the element. The right-column (normalized per revision) is calculated as: for each system, each element is normalized by the number of revisions for that system and the median across all systems is taken.

<table>
<thead>
<tr>
<th>Element</th>
<th>Normalized Per Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>rename</td>
<td>3.79</td>
</tr>
<tr>
<td>comment text</td>
<td>1.86</td>
</tr>
<tr>
<td>other text</td>
<td>0.78</td>
</tr>
<tr>
<td>literal value</td>
<td>0.56</td>
</tr>
<tr>
<td>name</td>
<td>0.32</td>
</tr>
<tr>
<td>expr</td>
<td>0.30</td>
</tr>
<tr>
<td>expr_stmt</td>
<td>0.19</td>
</tr>
<tr>
<td>call</td>
<td>0.17</td>
</tr>
<tr>
<td>operator</td>
<td>0.11</td>
</tr>
<tr>
<td>decl_stmt</td>
<td>0.11</td>
</tr>
<tr>
<td>parameter</td>
<td>0.10</td>
</tr>
<tr>
<td>comment</td>
<td>0.06</td>
</tr>
<tr>
<td>if</td>
<td>0.06</td>
</tr>
<tr>
<td>function</td>
<td>0.04</td>
</tr>
<tr>
<td>index</td>
<td>0.04</td>
</tr>
<tr>
<td>return</td>
<td>0.03</td>
</tr>
<tr>
<td>cpp:macro</td>
<td>0.03</td>
</tr>
<tr>
<td>else</td>
<td>0.02</td>
</tr>
<tr>
<td>function decl</td>
<td>0.02</td>
</tr>
<tr>
<td>cpp:value</td>
<td>0.01</td>
</tr>
</tbody>
</table>
slightly less frequent with 0.33 deleted and 0.35 inserted per revision on average (about one every three revisions on average). A declaration-statement was replaced with itself (appeared in both the delete and insert) almost once out of eight times on average (0.12). Lastly, functions were deleted as part of a replace 0.19 time per revision on average and inserted 0.14 times per revision on average. A function replaced itself (appeared in both the delete and insert) about once every twenty-five revisions on average.

This means that some expression-level replacements and statement-level expressions occur frequently, and that function replacements are not uncommon. We now turn to answering RQ1 and RQ2. For RQ1, code replacement at the text, expression, and statement level are frequently occurrences, and functions are not uncommonly replaced. The consequences of this (RQ2) are that replacement is sufficiently common enough at function, statement, and expression level that tree-edit distance algorithms will make inappropriate differences fairly often, which explains a major reason for their failure to arrive at a meaning difference and why they have not seen wide spread adoption.

9.4.2 Convert Results and Discussion

At least one type of code conversion (one transition), was present in all fifty systems. Table 9.12 and Table 9.13 show the total converts for each system (alphabetically). The left column gives the system, the middle column is the total number of conversions of all types that occurred over all revisions, and the last column normalizes the total number of conversion by the number of revisions. In total, there are almost 53,000 conversions between all systems, which averages to almost 12% of revisions containing a conversion (almost 1 in 8 revisions). As for individual systems, GamePlay had the most
conversions per revision (79.5%) and distortos has the smallest number of conversions per revision (3.4%). For the most part, however, the number of conversions per revisions is fairly consistent between systems. To confirm this, the number of revisions was plotted against the number of conversions. The results are shown in Figure 9.7. The x-axis is the number of revisions and the y-axis the total number of conversions. The resulting graph is fairly linear. The correlation between number of revisions and number of conversions is about 0.89, meaning that the number of conversions correlate highly with the number of revisions. From this we answer RQ3, concluding that conversions are not an uncommon event and that they are fairly consistent with the number of revisions.

In total, 35 different types of conversions occurred between the systems, each with a separate frequency. The results are shown in Table 9.14 and Table 9.15. Table 9.14 displays the frequency each type of transition occurred. The first column is the conversion with what it was converted from before the – and what it is converted to after the –. The middle column is the number times that conversion occurred in total across all systems. The last column is the percentage that conversion occurred out of all conversions. The most frequent by far are the conversions in the Statement category (82.1%).

Individually, conversions from expression-statements to declaration-statements are the most common being 34.8% of all conversions, and declaration-statement to expression-statement being the next most common at 24.5% of all conversions. After that, the percentage each type of conversion occurs drops quickly.

In order to adjust for systems with a large number of conversions, the data is also normalized by the total number of conversions that occurred in that system. The result is
Table 9.12. Total number of conversions for each system across all revisions. Left-column is system, middle is the number of converts, and the right column is the number of converts divided by revision (as percent). Last row contains total overall systems.

<table>
<thead>
<tr>
<th>System</th>
<th>Total Converts</th>
<th>Percent Converts Per Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>antimony</td>
<td>93</td>
<td>5.4%</td>
</tr>
<tr>
<td>appleseed</td>
<td>617</td>
<td>11.5%</td>
</tr>
<tr>
<td>BansheeEngine</td>
<td>823</td>
<td>26.9%</td>
</tr>
<tr>
<td>bitcoin</td>
<td>445</td>
<td>7.7%</td>
</tr>
<tr>
<td>blender</td>
<td>1,780</td>
<td>7.4%</td>
</tr>
<tr>
<td>cgal</td>
<td>4,889</td>
<td>12.4%</td>
</tr>
<tr>
<td>ChaiScript</td>
<td>261</td>
<td>19.9%</td>
</tr>
<tr>
<td>citra</td>
<td>262</td>
<td>9.0%</td>
</tr>
<tr>
<td>Clang</td>
<td>4,314</td>
<td>7.6%</td>
</tr>
<tr>
<td>cocos2d-x</td>
<td>2,015</td>
<td>14.5%</td>
</tr>
<tr>
<td>codeblocks</td>
<td>1,104</td>
<td>14.8%</td>
</tr>
<tr>
<td>codelite</td>
<td>2,550</td>
<td>32.2%</td>
</tr>
<tr>
<td>CRYENGINE</td>
<td>212</td>
<td>19.4%</td>
</tr>
<tr>
<td>CTK</td>
<td>256</td>
<td>9.1%</td>
</tr>
<tr>
<td>Dealii</td>
<td>1,026</td>
<td>5.0%</td>
</tr>
<tr>
<td>distortos</td>
<td>85</td>
<td>3.4%</td>
</tr>
<tr>
<td>Dlib</td>
<td>198</td>
<td>4.5%</td>
</tr>
<tr>
<td>engine</td>
<td>206</td>
<td>9.9%</td>
</tr>
<tr>
<td>folly</td>
<td>185</td>
<td>5.6%</td>
</tr>
<tr>
<td>GamePlay</td>
<td>1,077</td>
<td>79.5%</td>
</tr>
<tr>
<td>gnuradio</td>
<td>194</td>
<td>6.1%</td>
</tr>
<tr>
<td>griefly</td>
<td>54</td>
<td>5.0%</td>
</tr>
<tr>
<td>irods</td>
<td>925</td>
<td>32.3%</td>
</tr>
<tr>
<td>json</td>
<td>112</td>
<td>10.7%</td>
</tr>
<tr>
<td>kdevelop</td>
<td>2,299</td>
<td>13.8%</td>
</tr>
<tr>
<td>kokkos</td>
<td>80</td>
<td>10.7%</td>
</tr>
<tr>
<td>Mantella</td>
<td>67</td>
<td>5.1%</td>
</tr>
<tr>
<td>Mongo</td>
<td>3,261</td>
<td>12.4%</td>
</tr>
<tr>
<td>MultiMCS</td>
<td>176</td>
<td>13.9%</td>
</tr>
<tr>
<td>natron</td>
<td>2,535</td>
<td>25.2%</td>
</tr>
<tr>
<td>newton-dynamics</td>
<td>416</td>
<td>13.3%</td>
</tr>
<tr>
<td>nix</td>
<td>579</td>
<td>21.4%</td>
</tr>
<tr>
<td>ogre</td>
<td>1,647</td>
<td>31.2%</td>
</tr>
<tr>
<td>oito</td>
<td>245</td>
<td>10.0%</td>
</tr>
<tr>
<td>ola</td>
<td>386</td>
<td>8.6%</td>
</tr>
<tr>
<td>opencv</td>
<td>1,524</td>
<td>13.5%</td>
</tr>
<tr>
<td>openFrameworks</td>
<td>516</td>
<td>7.8%</td>
</tr>
</tbody>
</table>
Table 9.13. Total conversions (continued).

<table>
<thead>
<tr>
<th>Library</th>
<th>Total</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma</td>
<td>305</td>
<td>21.9%</td>
</tr>
<tr>
<td>ppsspp</td>
<td>1,507</td>
<td>9.9%</td>
</tr>
<tr>
<td>Qt</td>
<td>3,082</td>
<td>11.0%</td>
</tr>
<tr>
<td>QuantLib</td>
<td>744</td>
<td>8.7%</td>
</tr>
<tr>
<td>Rcpp</td>
<td>1,092</td>
<td>56.1%</td>
</tr>
<tr>
<td>rdkit</td>
<td>238</td>
<td>9.6%</td>
</tr>
<tr>
<td>stellarium</td>
<td>962</td>
<td>11.1%</td>
</tr>
<tr>
<td>supertux</td>
<td>278</td>
<td>6.5%</td>
</tr>
<tr>
<td>swift</td>
<td>3,424</td>
<td>11.3%</td>
</tr>
<tr>
<td>tira</td>
<td>54</td>
<td>6.5%</td>
</tr>
<tr>
<td>tfs-old-svn</td>
<td>596</td>
<td>12.8%</td>
</tr>
<tr>
<td>Urho3D</td>
<td>1,020</td>
<td>19.5%</td>
</tr>
<tr>
<td>xbmc</td>
<td>2,250</td>
<td>8.9%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>52,966</td>
<td>11.9%</td>
</tr>
</tbody>
</table>

Figure 9.7. The total number of revisions plotted against the number of conversions. X-axis is number of revisions and y-axis is the number of conversions. The result is fairly linear.
Table 9.14. Frequency of each time of conversion. The left column is the number of conversions, the middle column is the raw number of how many times the conversion occurred between all systems, and the right column is the percentage that conversion occurred out of all types.

<table>
<thead>
<tr>
<th>Conversion</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expr_stmt-decl_stmt</code></td>
<td>18,432</td>
<td>34.8%</td>
</tr>
<tr>
<td><code>decl_stmt-expr_stmt</code></td>
<td>12,976</td>
<td>24.5%</td>
</tr>
<tr>
<td><code>expr_stmt-return</code></td>
<td>3,652</td>
<td>6.9%</td>
</tr>
<tr>
<td><code>return-expr_stmt</code></td>
<td>3,332</td>
<td>6.3%</td>
</tr>
<tr>
<td><code>return-decl_stmt</code></td>
<td>3,317</td>
<td>6.3%</td>
</tr>
<tr>
<td><code>decl_stmt-return</code></td>
<td>1,775</td>
<td>3.4%</td>
</tr>
<tr>
<td><code>else-elseif</code></td>
<td>1,588</td>
<td>3.0%</td>
</tr>
<tr>
<td><code>private-public</code></td>
<td>1,067</td>
<td>2.0%</td>
</tr>
<tr>
<td><code>public-private</code></td>
<td>911</td>
<td>1.7%</td>
</tr>
<tr>
<td><code>struct-class</code></td>
<td>893</td>
<td>1.7%</td>
</tr>
<tr>
<td><code>private-protected</code></td>
<td>693</td>
<td>1.3%</td>
</tr>
<tr>
<td><code>elseif-else</code></td>
<td>671</td>
<td>1.3%</td>
</tr>
<tr>
<td><code>while-for</code></td>
<td>590</td>
<td>1.1%</td>
</tr>
<tr>
<td><code>class-struct</code></td>
<td>589</td>
<td>1.1%</td>
</tr>
<tr>
<td><code>for-while</code></td>
<td>335</td>
<td>0.6%</td>
</tr>
<tr>
<td><code>foreach-for</code></td>
<td>306</td>
<td>0.6%</td>
</tr>
<tr>
<td><code>protected-private</code></td>
<td>266</td>
<td>0.5%</td>
</tr>
<tr>
<td><code>protected-public</code></td>
<td>244</td>
<td>0.5%</td>
</tr>
<tr>
<td><code>for-foreach</code></td>
<td>237</td>
<td>0.4%</td>
</tr>
<tr>
<td><code>if-while</code></td>
<td>212</td>
<td>0.4%</td>
</tr>
<tr>
<td><code>if-for</code></td>
<td>206</td>
<td>0.4%</td>
</tr>
<tr>
<td><code>for-if</code></td>
<td>198</td>
<td>0.4%</td>
</tr>
<tr>
<td><code>public-protected</code></td>
<td>175</td>
<td>0.3%</td>
</tr>
<tr>
<td><code>while-if</code></td>
<td>122</td>
<td>0.2%</td>
</tr>
<tr>
<td><code>class-enum</code></td>
<td>48</td>
<td>0.1%</td>
</tr>
<tr>
<td><code>struct-enum</code></td>
<td>46</td>
<td>0.1%</td>
</tr>
<tr>
<td><code>union-struct</code></td>
<td>24</td>
<td>0.0%</td>
</tr>
<tr>
<td><code>while-foreach</code></td>
<td>21</td>
<td>0.0%</td>
</tr>
<tr>
<td><code>struct-union</code></td>
<td>14</td>
<td>0.0%</td>
</tr>
<tr>
<td><code>class-union</code></td>
<td>5</td>
<td>0.0%</td>
</tr>
<tr>
<td><code>enum-struct</code></td>
<td>5</td>
<td>0.0%</td>
</tr>
<tr>
<td><code>foreach-if</code></td>
<td>5</td>
<td>0.0%</td>
</tr>
<tr>
<td><code>if-foreach</code></td>
<td>5</td>
<td>0.0%</td>
</tr>
<tr>
<td><code>enum-class</code></td>
<td>3</td>
<td>0.0%</td>
</tr>
<tr>
<td><code>foreach-while</code></td>
<td>3</td>
<td>0.0%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>52,966</td>
<td>100.0%</td>
</tr>
</tbody>
</table>
Table 9.15. Percentage of each type of conversion normalized for each system by the total number of conversions.

<table>
<thead>
<tr>
<th>Conversion</th>
<th>Normalized Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr_stmt-decl_stmt</td>
<td>31.4%</td>
</tr>
<tr>
<td>decl_stmt-expr_stmt</td>
<td>23.1%</td>
</tr>
<tr>
<td>expr_stmt-return</td>
<td>5.3%</td>
</tr>
<tr>
<td>return-expr_stmt</td>
<td>4.7%</td>
</tr>
<tr>
<td>return-decl_stmt</td>
<td>4.6%</td>
</tr>
<tr>
<td>decl_stmt-return</td>
<td>2.5%</td>
</tr>
<tr>
<td>else-elseif</td>
<td>2.3%</td>
</tr>
<tr>
<td>struct-class</td>
<td>1.1%</td>
</tr>
<tr>
<td>elseif-else</td>
<td>1.1%</td>
</tr>
<tr>
<td>private-public</td>
<td>1.0%</td>
</tr>
<tr>
<td>private-protected</td>
<td>0.8%</td>
</tr>
<tr>
<td>while-for</td>
<td>0.8%</td>
</tr>
<tr>
<td>public-private</td>
<td>0.6%</td>
</tr>
<tr>
<td>class-struct</td>
<td>0.6%</td>
</tr>
<tr>
<td>for-while</td>
<td>0.4%</td>
</tr>
<tr>
<td>protected-private</td>
<td>0.4%</td>
</tr>
<tr>
<td>if-while</td>
<td>0.3%</td>
</tr>
<tr>
<td>if-for</td>
<td>0.3%</td>
</tr>
<tr>
<td>for-if</td>
<td>0.2%</td>
</tr>
<tr>
<td>while-if</td>
<td>0.2%</td>
</tr>
<tr>
<td>protected-public</td>
<td>0.1%</td>
</tr>
<tr>
<td>public-protected</td>
<td>0.1%</td>
</tr>
<tr>
<td>class-enum</td>
<td>0.0%</td>
</tr>
<tr>
<td>struct-enum</td>
<td>0.0%</td>
</tr>
<tr>
<td>for-foreach</td>
<td>0.0%</td>
</tr>
<tr>
<td>foreach-for</td>
<td>0.0%</td>
</tr>
<tr>
<td>struct-union</td>
<td>0.0%</td>
</tr>
<tr>
<td>union-struct</td>
<td>0.0%</td>
</tr>
<tr>
<td>class-union</td>
<td>0.0%</td>
</tr>
<tr>
<td>enum-class</td>
<td>0.0%</td>
</tr>
<tr>
<td>enum-struct</td>
<td>0.0%</td>
</tr>
<tr>
<td>foreach-while</td>
<td>0.0%</td>
</tr>
<tr>
<td>while-foreach</td>
<td>0.0%</td>
</tr>
<tr>
<td>foreach-if</td>
<td>0.0%</td>
</tr>
<tr>
<td>if-foreach</td>
<td>0.0%</td>
</tr>
</tbody>
</table>
in Table 9.15. The first column gives the conversion, and the second column is calculated as follows. First, the percentage each conversion occurred for a system is computed, then the median between all systems is taken. Normalization does not have a major effect on the data, meaning that the distribution per system is fairly consistent.

In conclusion, conversion occurs on average once every eight revisions, and 82% of the time it is between declaration-statements, expression-statements, and return-statements. The consequences (RQ4), is that the frequency for which conversions occur justify the necessity to have such detection in a syntactic differencer, but the biggest gain and most often required is in the Statement category. A lack of support for conversion, will lead to fairly frequent incorrect difference reporting that is also difficult to detect without specific markup.

9.5 Threats to Validity

The following threats to validity are identified with the study.

First, srcDiff, srcML, Git, and other tooling used as part of the study are complicated pieces of software, and as such contain faults. The parsing of C++ code is notoriously difficult, in part, due to preprocessor statements making C++ a non-context free grammar. The extent to which the tools report inappropriate ASTs is not known, yet srcML has a rigorous test suite and srcDiff is also well tested. Despite this, a negligible number of revisions failed to report a correct XML document (possible due to program crash, etc.). A negligible number of commits will not affect the vast amount of data collected.
Only standard C++ extensions were used to locate C++ files. Most of the projects used only these extensions, however, some used unusual extensions which possibly contained C++ code. For example, a C++ extension followed by `.in`, which would possible denote a file that is used to generate code as part of the build system, and `.tpp`, which could possible contain template code. These were only noticed in a few of the repositories. For future work, these files can be examined more closely and a determination made as to whether they should be included as part of the data collected.

The `.h` extension was treated as a C++, however, it is used both as a C and C++ header extension. Some of the projects included both C and C++ code and used `.h` for both. As C++ is largely a super-set of C++ (with a few exceptions), the parsing of C code as C++ code is a minor threat.

srcML provides support for the markup of Qt extensions. Currently, these are always on. Some code may have been incorrectly converted to srcML when a non-Qt project used an identifier that is part of the Qt extensions. In the future, it is planned for srcML to support Qt extensions optionally.

Some of the projects have committed code that was auto-generated by some tooling. Auto-generated files may exist throughout the history of a project or may have appeared and disappeared at any point in the history of the software. For the most part, we are uninterested in changes to auto-generated code, however, as they can appear at any point in history, they may be difficult to find. Investigation into auto-detection of all generated files through the history of projects can be a valid research topic. With the systems used as part of the study, two instances of auto-generated files were found. One was in `ppsspp`,
and the other in codelite. They were found due to srcDiff’s current difficulty in handling files with, for example, large static arrays and string literals, as they result in elements with exceedingly large numbers of children and text nodes for which Myers’ algorithm O(ND) algorithm [Myers 1986] is not quick enough. In these two cases, the file list generation part of the data collection was instructed to ignore these auto-generated files.

Lastly, only C++ was used as part of the study. Results may be somewhat different for other languages. Although, we have experimented a little with conversion on Java (look at a 100 random revisions of a few systems) and have found a consistent amount of function conversions. Future work applying the approach to other languages may be worthwhile.

9.6 Conclusion

An investigation of fifty software systems to determine the prevalence and consequences of code replace and code conversion is performed. The results are that code replace is a frequently occurring event at the text, expression, and statement level, and not uncommon for functions (RQ1). Code conversion is also a sufficiently common event primarily between declaration-statements, expression-statements, and return statements (RQ3).

The consequences of both (RQ2 and RQ4) are that support for both are needed in a syntactic differencer, and that differencers that solely use tree-edit distance algorithms will often arrive at an incorrect difference. That is, code replacement and code conversion are a major reason why srcDiff outperforms current syntactical differencing approaches.
CHAPTER 10

Conclusion

In this dissertation, the syntactic-differencing approach, srcDiff, is presented along with two evaluations. In addition, the syntactic-differencing approach is applied to two software engineering research problems: investigation into changes in method stereotype and investigation into code replace and code conversion.

In total, ten research questions (4 are evaluations and 6 for the two studies using srcDiff) are proposed and answered. The research questions are repeated here and answered, separately for the evaluations and software engineering research applications.

• **Evaluation RQ1**: Does srcDiff produce changes that are easier to read and more understandable than the other approaches?

• **Evaluation RQ2**: Is srcDiff preferred over the other approaches?

• **Evaluation RQ3**: Does srcDiff result in a delta that is more accurate than the other approaches?

• **Evaluation RQ4**: Is srcDiff scalable and practical to use in a software development environment?

The study shows that srcDiff is overwhelming chosen over GumTree [Falleri et al. 2014], GitHub’s difference view [GitHub 2016], and Mergely [Peabody 2016] as the easiest to read and understand (**Evaluation RQ1** and the most preferable (**Evaluation RQ2**). With all participants, srcDiff is about equally as accurate as GumTree. With experts, srcDiff is overwhelmingly considered the most accurate (**Evaluation RQ3**). The
timing comparison in CHAPTER 7, shows that srcDiff has a quicker execution time than GumTree, and the applications of srcDiff to full history of many software projects (CHAPTER 8 and CHAPTER 9) shows that srcDiff is highly scalable and practical to use in a software development environment (Evaluation RQ4).

- **Stereotype RQ1**: How stable are a method’s stereotype?
- **Stereotype RQ2**: What consequences are there to a method’s stereotype changing?

The results of CHAPTER 8 conclude that method stereotypes are highly stable (Stereotype RQ1) and that certain types of method stereotypes can indicate poor design or inappropriate function changes (Stereotype RQ2).

- **Replace RQ1**: How commonly do developers replace and rewrite code?
- **Replace RQ2**: What are the consequences of replace and rewriting code?

The results of CHAPTER 9 on replace show that replace occurs extremely frequently on average at the text level, and frequently at the expression and statement level. In addition, function replacements are not uncommon (Replace RQ1). The consequences of replaced code and its prevalence gives a measure to how much and how often srcDiff can outperform tree-edit distance based syntactic differencers. Replace code must be handled by a syntactic difference to arrive at meaningful differences (Replace RQ2).

- **Convert RQ1**: How often do developers convert code from one type to another?
- **Convert RQ2**: What are the consequence of converting code?
The results of CHAPTER 9 for converts show that converts are fairly common, but not as common as code replace (Convert RQ1). The consequences are analogous to code replace. Code conversion occurs frequently enough that differencers must support code conversion detection and markup. Without support, differencers will very frequently report poor differences (Convert RQ2).

srcDiff is beyond a research prototype, however, it is constantly being improved through the addition and refinement of rules, as well as, other steps to the algorithm. For example, an edit script correction step has been added after the use of Myers’ algorithm [Myers 1986] to avoid large structures and statements from being reported as inserted and deleted because they have a small amount of change and a much smaller entity was reported as common between them.

Future work will consist of application of srcDiff to various other software engineering tasks. Since srcDiff boasts greater accuracy, one avenue of research is repeating former research tasks that used other syntactic differencers and seeing if better results are achieved. Other future work consists of investigating what developer’s consider a move and adding extended support to srcDiff for move detection, and an investigation into software merging.
APPENDIX A

Code for srcDiff Tool Integration into Git

The following is bash script for integrating the srcDiff tool into a Git repository.

```bash
#!/bin/sh

if [[ $2 = "/dev/null" ]]; then
    touch srcdiff_temp$(bash -c "echo $5 | grep -o './[^.]*$'")
    [ $# -eq 7 ] && srcdiff --unified=3 srcdiff_temp$(bash -c "echo $5 | grep -o './[^.]*$'") "$5"
    rm -f srcdiff_temp$(bash -c "echo $5 | grep -o './[^.]*$'")
elif [[ $5 = "/dev/null" ]]; then
    touch srcdiff_temp$(bash -c "echo $2 | grep -o './[^.]*$'")
    [ $# -eq 7 ] && srcdiff --unified=3 "$2" srcdiff_temp$(bash -c "echo $2 | grep -o './[^.]*$'")
    rm -f srcdiff_temp$(bash -c "echo $2 | grep -o './[^.]*$'")
else
    [ $# -eq 7 ] && srcdiff --unified=3 "$2" "$5"
fi
```
APPENDIX B

Complete srcDiff Tool Options

The following is a complete list of the srcDiff Tool options as produced by the --help option.

Usage: srcdiff [options] <original_src_infile modified_src_infile>... [-o <srcDiff_outfile>]

Translates C, C++, and Java source code into the XML source-code representation srcDiff.
Multiple files are stored in a srcDiff archive.

General Options:
- h [ --help ] Output srcdiff help message
- V [ --version ] Output srcdiff version
- o [ --output ] arg (=) Specify output filename
- z [ --compress ] Compress the output
- v [ --verbose ] Verbose messaging
- q [ --quiet ] Silence messaging

Input Options:
--files-from arg Set the input to be a list of file pairs from the provided file. Pairs are of the format: original|modified
--git arg Input from a Subversion repository. Example: --git http://example.org@HASH

srcml Options:
- n [ --archive ] Output srcDiff as an archive
- t [ --src-encoding ] arg (=ISO-8859-1) Set the input source encoding
- x [ --xml-encoding ] arg (=UTF-8) Set the output XML encoding
- l [ --language ] arg (=C++) Set the input source programming language
- f [ --filename ] arg Override unit filename
--register-ext arg Register an extension to language pair to be used during parsing
--url arg Set the root url attribute
--s [ --src-version ] arg Set the root version attribute
--xmlns arg Set the prefix associationed with a namespace or register a new one. of the form --xmlns:prefix=url or --xmlns=url for default prefix.
--position Output additional position information on the srcML elements
--tabs arg (=8) Set the tabstop size
--no-xml-decl Do not output the xml declaration
--cpp-markup-else Markup up #else contents (default)
--cpp-text-else
Do not markup #else contents
--cpp-markup-if0
Markup up #if 0 contents
--cpp-text-if0
Do not markup #if 0 contents (default)

srcdiff Options:
-m [ --method ] arg (=collect,group-diff)
Set srcdiff parsing method
--burst
Output each input file to a single srcDiff document. -o gives output directory
--srcml
Also, output the original and modified srcML of each file when burst enabled
-u [ --unified ] [=arg(=3)]
Output as colorized unified diff with provided context. Number is lines of context, 'all' or -1 for entire file, 'function' for encompassing function (default = 3)
-y [ --side-by-side ] [=arg(=7)]
Output as colorized side-by-side diff
--html
Output unified/side-by-side view in html
-W [ --ignore-all-space ]
Ignore all whitespace when outputting unified/side-by-side view
-w [ --ignore-space ]
Ignore whitespace when outputting unified/side-by-side view
-c [ --ignore-comments ]
Ignore comments when outputting unified/side-by-side view
--highlight [=arg(=partial)] (=partial)
Syntax-highlighting for unified/side-by-side view. None, partial (default), or full
--theme arg (=default)
Select theme for syntax-highlighting. Default or monokai
--summary [=arg(=text)]
Output a summary of the differences. Options 'text' and/or 'table' summary. Default 'text'

Report bugs to mdecker6@kent.edu
APPENDIX C

Source-code Changes Used in Preliminary Study

This contains the source code change example and samples used in the preliminary study. For each, the original and modified code, GNU diff, and srcDiff are provided each as a separate image. In the actual study, the original and modified were side-by-side, but as we are not able to do so for all samples, we place them one after the other. Even one after the other, some images may be altered to fit. The complete survey is available at sdml.cs.kent.edu/srcDiff/dissertation/preliminary_study as part of the replication package.

Example Source Original

<table>
<thead>
<tr>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo * bar = 0;</td>
</tr>
</tbody>
</table>

Example Source Modified

<table>
<thead>
<tr>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo * bar = NULL;</td>
</tr>
</tbody>
</table>
Example GNU `diff`

```
diff
-foo * bar = 0;
+foo * bar = NULL;
```

Example srcDiff

```
srcDiff
foo * bar = 0NULL;
```
Sample 1 Source Original

<table>
<thead>
<tr>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>style.addProperty(&quot;fo:font-weight&quot;, boldness * 10, KoGenStyle::TextType);</td>
</tr>
</tbody>
</table>

Sample 1 Source Modified

<table>
<thead>
<tr>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>style.addProperty(&quot;fo:font-weight&quot;, qBound(10, boldness, 90) * 10, KoGenStyle::TextType);</td>
</tr>
</tbody>
</table>

Sample 1 GNU diff

<table>
<thead>
<tr>
<th>diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>-style.addProperty(&quot;fo:font-weight&quot;, boldness * 10, KoGenStyle::TextType);</td>
</tr>
<tr>
<td>+style.addProperty(&quot;fo:font-weight&quot;, qBound(10, boldness, 90) * 10, KoGenStyle::TextType);</td>
</tr>
</tbody>
</table>

Sample 1 srcDiff

<table>
<thead>
<tr>
<th>srcDiff</th>
</tr>
</thead>
<tbody>
<tr>
<td>style.addProperty(&quot;fo:font-weight&quot;, qBound(10, boldness, 90) * 10, KoGenStyle::TextType);</td>
</tr>
</tbody>
</table>
Sample 2 Source Original

```c
Original

d->fallBackColorSpace = _fallBackToCs->dstColorSpace();
```

Sample 2 Source Modified

```c
Modified

d->fallBackColorSpace = _fallBackToCs->srcColorSpace();
```

Sample 2 GNU diff

```diff
diff

-d->fallBackColorSpace = _fallBackToCs->dstColorSpace();
+\n\n+\n\n+\n\n+\n\n+\n\n+\n\n+\n\n+\n```

Sample 2 srcDiff

```c
srcDiff

d->fallBackColorSpace = _fallBackToCs->\textcolor{red}{dstColorSpace}\textcolor{green}{srcColorSpace}();
```
Sample 3 Source Original

```c
/* This file is part of the KDE project
 * Copyright (C) 2008 Fredy Yanardi <fyanardi@gmail.com>
 * Copyright (C) 2008 Thorsten Zachmann <zachmann@kde.org>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public License
 * along with this library; see the file COPYING.LIB. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
 * Boston, MA 02110-1301, USA.
 */
```

Sample 3 Source Modified

```c
/* This file is part of the KDE project
 * Copyright (C) 2008 Fredy Yanardi <fyanardi@gmail.com>
 * Copyright (C) 2008-2009 Thorsten Zachmann <zachmann@kde.org>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public License
 * along with this library; see the file COPYING.LIB. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
 * Boston, MA 02110-1301, USA.
 */
```
Sample 3 GNU diff

diff

/* This file is part of the KDE project
 * Copyright (C) 2008 Fredy Yanardi <fyanardi@gmail.com>
 * Copyright (C) 2008 Thorsten Zachmann <zachmann@kde.org>
 + Copyright (C) 2008-2009 Thorsten Zachmann <zachmann#kde.org>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public License
 * along with this library; see the file COPYING.LIB. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
 * Boston, MA 02110-1301, USA.
 */

Sample 3 srcDiff

srcDiff

/* This file is part of the KDE project
 * Copyright (C) 2008 Fredy Yanardi <fyanardi@gmail.com>
 + Copyright (C) 20082008-2009 Thorsten Zachmann <zachmann@kde.org>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public License
 * along with this library; see the file COPYING.LIB. If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
 * Boston, MA 02110-1301, USA.
 */
Sample 4 Source Original

```cpp
if (1numberedParagraph && e.tagName() != "list-item" && !listHeader)
    continue;

if (!firstTime) {
    // use empty formats to not inherit from the prev parag
    QTextBlockFormat bf;
    QTextCharFormat cf;
    cursor.insertBlock(bf, cf);
} else {
    firstTime = false;
}
QTextBlock current = cursor.block();
```

Sample 4 Source Modified

```cpp
if (1numberedParagraph && e.tagName() != "list-item" && !listHeader)
    continue;

if (!firstTime && !numberedParagraph) {
    // use empty formats to not inherit from the prev parag
    QTextBlockFormat bf;
    QTextCharFormat cf;
    cursor.insertBlock(bf, cf);
}
firstTime = false;
QTextBlock current = cursor.block();
```
Sample 4 GNU diff

```cpp
if (!numberedParagraph && e.tagName() != "list-item" && !listHeader)
    continue;

- if (!firstTime) {
+ if (!firstTime && !numberedParagraph) {
    // use empty formats to not inherit from the prev parag
    QTextBlockFormat bf;
    QTextCharFormat cf;
    cursor.insertBlock(bf, cf);
    
} else {
-    firstTime = false;
} 
+firstTime = false;
QTextBlock current = cursor.block();
```

Sample 4 srcDiff

```cpp
if (!numberedParagraph && e.tagName() != "list-item" && !listHeader)
    continue;

if (!firstTime && !numberedParagraph) {
    // use empty formats to not inherit from the prev parag
    QTextBlockFormat bf;
    QTextCharFormat cf;
    cursor.insertBlock(bf, cf);
} else {

    firstTime = false;
}

firstTime = false;
QTextBlock current = cursor.block();
```
Sample 5 Source Original

```
KoView* KoMainWindow::currentView() const
{
    // XXX
    if (d->activeView) {
        return d->activeView;
    }
    else {
        return d->rootViews.first();
    }
}
```

Sample 5 Source Modified

```
KoView* KoMainWindow::currentView() const
{
    // XXX
    if (d->activeView) {
        return d->activeView;
    }
    else if (!d->rootViews.isEmpty()) {
        return d->rootViews.first();
    }
    return 0;
}
```
**Sample 5 GNU diff**

```
KoView* KoMainWindow::currentView() const
{
    // XXX
    if (d->activeView) {
        return d->activeView;
    }
    else {
        else if (!d->rootViews.isEmpty()) {
            return d->rootViews.first();
        }
        return 0;
    }
}
```

**Sample 5 srcDiff**

```
KoView* KoMainWindow::currentView() const
{
    // XXX
    if (d->activeView) {
        return d->activeView;
    }
    else if (!d->rootViews.isEmpty()) {
        return d->rootViews.first();
    }
    return 0;::<j>
}
```
Sample 6 Source Original

Original

```cpp
if (m_textShapeData->endPosition() < 0) { // not layouted yet.
    if (lay == 0)
        kWarning(32500) << "Painting shape that doesn't have a kotext doc-layout, which can't work";
    else if (!lay->hasLayouter())
        lay->setLayout(new Layout(lay));
    return;
}
```

Sample 6 Source Modified

Modified

```cpp
if (m_textShapeData->endPosition() < 0) { // not layouted yet.
    if (lay == 0) {
        kWarning(32500) << "Painting shape that doesn't have a kotext doc-layout, which can't work";
        return;
    } else if (!lay->hasLayouter()) {
        lay->setLayout(new Layout(lay));
    } if (!m_pageProvider) {
        return;
    }
```
Sample 6 GNU diff

diff

if (m_textShapeData->endPosition() < 0) { // not layouted yet.
  - if (lay == 0)
  + if (lay == 0) {
      kWarning(32500) << "Painting shape that doesn't have a 
                    kotext doc-layout, which can't work";
  - else if (! lay->hasLayouter())
  +    return;
  + }
  + else if (! lay->hasLayouter()) {
    lay->setLayout(new Layout(lay));
  - return;
  + }
  + if (!m_pageProvider) {
  +    return;
  + }
}
Sample 6 srcDiff

```
srcDiff

if (m_textShapeData->endPosition() < 0) { // not layouted yet.
    if (lay == 0) {
        kWarning(32500) << "Painting shape that doesn't have a kotext doc-layout, which can't work";
        return;
    }
    else if (!lay->hasLayouter()) {
        lay->setLayout(new Layout(lay));
    }
    if (!m_pageProvider) {
        return;
    }
}
```
Sample 7 Source Original

```
QWidget* KisToolSelectPath::optionWidget()
{
    return m_optWidget;
}
```

Sample 7 Source Modified

```
QMap<QString, QWidget *> KisToolSelectPath::createOptionWidgets()
{
    QMap<QString, QWidget *> map = KoCreatePathTool::createOptionWidgets();
    map.insert(i18n("Tool Options"), createOptionWidget());
    return map;
}
```

Sample 7 GNU diff

```
- QWidget* KisToolSelectPath::optionWidget()
+ QMap<QString, QWidget *> KisToolSelectPath::createOptionWidgets()
     {
         return m_optWidget;
+        QMap<QString, QWidget *> map = KoCreatePathTool::createOptionWidgets();
+        map.insert(i18n("Tool Options"), createOptionWidget());
+        return map;
     }
```
Sample 7 srcDiff

```cpp
QWidget* KisToolSelectPath::optionWidget()
{
    return m_optWidget;
}
QMap<QString, QWidget *> KisToolSelectPath::createOptionWidgets()
{
    QMap<QString, QWidget *> map = KoCreatePathTool::createOptionWidgets();
    map.insert(i18n("Tool Options"), createOptionWidget());
    return map;
}
```
void MainWindow::slotCloseAllViews()
{
    // Attention: Very touchy code... you know what you're doing? Gooooood ;) 
    d->forQuit = true;
    if (queryClose()) { 
        // In case the document is embedded we close all open "extra-shells"
        if (d->rootDoc && d->rootDoc->isEmbedded()) {
            hide();
            d->rootDoc->removeShell(this);
            QList<MainWindow*> shells = d->rootDoc->shells();
            while (!shells.isEmpty()) {
                MainWindow* window = shells.takeFirst();
                hide();
                delete window;
                d->rootDoc = 0;
            }
        } // not embedded -> destroy the document and all shells/views ;)
    } else {
        setRootDocument (nullptr);
    }
    close(); // close this window (and quit the app if necessary)
}

d->forQuit = false;
void KoMainWindow::slotCloseAllViews()
{
    d->forQuit = true;
    if (queryClose()) {
        hide();
        d->rootDoc->removeShell(this);
        QList<KoMainWindows*> shells = d->rootDoc->shells();
        d->rootDoc = 0;
        while (!shells.isEmpty()) {
            KoMainWindows* window = shells.takeFirst();
            window->hide();
            delete window;
        }
        close(); // close this window (and quit the app if necessary)
    }
    d->forQuit = false;
}
void KoMainWindow::slotCloseAllViews()
{
    // Attention: Very touchy code... you know what you're doing? Gooooood :)
    d->forQuit = true;
    if (queryClose()) {
        // In case the document is embedded we close all open "extra-shells"
        if (d->rootDoc && d->rootDoc->isEmbedded()) {
            hide();
            d->rootDoc->removeShell(this);
            QList<KoMainWindow*> shells = d->rootDoc->shells();
            while (!shells.isEmpty()) {
                KoMainWindow* window = shells.takeFirst();
                window->hide();
                delete window;
                d->rootDoc = 0;
            }
            // not embedded -> destroy the document and all shells/views ;)
            else {
                setRootDocument(0);
                hide();
                d->rootDoc->removeShell(this);
                QList<KoMainWindow*> shells = d->rootDoc->shells();
                d->rootDoc = 0;
                while (!shells.isEmpty()) {
                    KoMainWindow* window = shells.takeFirst();
                    window->hide();
                    delete window;
                }
                close(); // close this window (and quit the app if necessary)
            }
            d->forQuit = false;
        }
    }
}
void KoMainWindow::slotCloseAllWindows()
{
  // Attention: Very touchy code... you know what you're doing? Gooooood :) <
  d->forQuit = true;
  if (queryClose()) {
    // In case the document is embedded we close all open "extra-shells"
    if (d->rootDoc && d->rootDoc->isEmbedded()) {
      hide();
      d->rootDoc->removeShell(this);
      QList<KoMainWindow*> shells = d->rootDoc->shells();
      d->rootDoc = 0; <
      while (!shells.isEmpty()) {
        KoMainWindow* window = shells.takeFirst();
        window->hide();
        delete window;
        d->rootDoc = 0; <
      }
    } <
    // not embedded -> destroy the document and all shells/views <
    else {
      setRootDocument(0); <
      close(); // close this window (and quit the app if necessary)
    }
    d->forQuit = false;
}
**Sample 9 Source Original**

<table>
<thead>
<tr>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (shape-&gt;hasAdditionalAttribute(&quot;text:anchor-type&quot;)) {</td>
</tr>
<tr>
<td>QString anchorType = shape-&gt;additionalAttribute(&quot;text:anchor-type&quot;);</td>
</tr>
<tr>
<td>// page anchored shapes are handled differently</td>
</tr>
<tr>
<td>if (anchorType != &quot;page&quot;) {</td>
</tr>
<tr>
<td>KoTextAnchor *anchor = new KoTextAnchor(shape);</td>
</tr>
<tr>
<td>anchor-&gt;loadOffFromShape(element);</td>
</tr>
<tr>
<td>d-&gt;textSharedData-&gt;shapeInserted(shape, element, d-&gt;context);</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>KoTextDocumentLayout *layout</td>
</tr>
<tr>
<td>= qobject_cast&lt;KoTextDocumentLayout*&gt;(cursor.block().document()-&gt;documentLayout());</td>
</tr>
<tr>
<td>if (layout) {</td>
</tr>
<tr>
<td>KoInlineTextObjectManager *textObjectManager = layout-&gt;inlineTextObjectManager();</td>
</tr>
<tr>
<td>if (textObjectManager) {</td>
</tr>
<tr>
<td>textObjectManager-&gt;insertInlineObject(cursor, anchor);</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>else {</td>
</tr>
<tr>
<td>d-&gt;textSharedData-&gt;shapeInserted(shape, element, d-&gt;context);</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

**Sample 9 Source Modified**

<table>
<thead>
<tr>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>QString anchorType;</td>
</tr>
<tr>
<td>if (shape-&gt;hasAdditionalAttribute(&quot;text:anchor-type&quot;)) {</td>
</tr>
<tr>
<td>anchorType = shape-&gt;additionalAttribute(&quot;text:anchor-type&quot;);</td>
</tr>
<tr>
<td>} else if (element.hasAttributeNS(KoXmlNS::text, &quot;anchor-type&quot;)) {</td>
</tr>
<tr>
<td>anchorType = element.attributeNS(KoXmlNS::text, &quot;anchor-type&quot;);</td>
</tr>
<tr>
<td>else {</td>
</tr>
<tr>
<td>anchorType = &quot;as-char&quot;; // default value</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>// page anchored shapes are handled differently</td>
</tr>
<tr>
<td>if (anchorType != &quot;page&quot;) {</td>
</tr>
<tr>
<td>KoTextAnchor *anchor = new KoTextAnchor(shape);</td>
</tr>
<tr>
<td>anchor-&gt;loadOffFromShape(element);</td>
</tr>
<tr>
<td>d-&gt;textSharedData-&gt;shapeInserted(shape, element, d-&gt;context);</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>KoTextDocumentLayout *layout</td>
</tr>
<tr>
<td>= qobject_cast&lt;KoTextDocumentLayout*&gt;(cursor.block().document()-&gt;documentLayout());</td>
</tr>
<tr>
<td>if (layout) {</td>
</tr>
<tr>
<td>KoInlineTextObjectManager *textObjectManager = layout-&gt;inlineTextObjectManager();</td>
</tr>
<tr>
<td>if (textObjectManager) {</td>
</tr>
<tr>
<td>textObjectManager-&gt;insertInlineObject(cursor, anchor);</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>else {</td>
</tr>
<tr>
<td>d-&gt;textSharedData-&gt;shapeInserted(shape, element, d-&gt;context);</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
Sample 9 GNU diff

```cpp
diff
-if (shape->hasAdditionalAttribute("text:anchor-type")) {
-  QString anchorType = shape->additionalAttribute("text:anchor-type");
-  // page anchored shapes are handled differently
-  if (anchorType == "page") {
-    KoTextAnchor *anchor = new KoTextAnchor(shape);
-    anchor->loadOdfFromShape(element);
-    d->textSharedData->shapeInserted(shape, element, d->context);
-  }
+QString anchorType;
+if (shape->hasAdditionalAttribute("text:anchor-type"))
+  anchorType = shape->additionalAttribute("text:anchor-type");
+else if (element.hasAttributeNS(KoXmlNS::text, "anchor-type"))
+  anchorType = element.attributeNS(KoXmlNS::text, "anchor-type");
+else
+  anchorType = "as-char"; // default value
+
-  KoTextDocumentLayout *layout
-  = qobject_cast<KoTextDocumentLayout*>(cursor.block().document()->documentLayout());
-  if (layout) {
-    KoInlineTextObjectManager *textObjectManager = layout->inlineTextObjectManager();
-    if (textObjectManager) {
-      textObjectManager->insertInlineObject(cursor, anchor);
-    }
-  }
+// page anchored shapes are handled differently
+if (anchorType == "page") {
+  KoTextAnchor *anchor = new KoTextAnchor(shape);
+  anchor->loadOdfFromShape(element);
+  d->textSharedData->shapeInserted(shape, element, d->context);
+}
+KoTextDocumentLayout *layout
+  = qobject_cast<KoTextDocumentLayout*>(cursor.block().document()->documentLayout());
+if (layout) {
+  KoInlineTextObjectManager *textObjectManager = layout->inlineTextObjectManager();
+  if (textObjectManager) {
+    textObjectManager->insertInlineObject(cursor, anchor);
+  }
-  else {
-    d->textSharedData->shapeInserted(shape, element, d->context);
-  }
+  } else {
+    d->textSharedData->shapeInserted(shape, element, d->context);
+  }
```
```cpp
srcDiff

if (shape->hasAdditionalAttribute("text:anchor-type")) {
  QString anchorType = shape->additionalAttribute("text:anchor-type");
  if (shape->hasAdditionalAttribute("text:anchor-type"))
    anchorType = shape->additionalAttribute("text:anchor-type");
  else if (element.hasAttributeNS(KoXmlNS::text, "anchor-type")]
    anchorType = element.attributeNS(KoXmlNS::text, "anchor-type");
  else
    anchorType = "as-char"; // default value

  // page anchored shapes are handled differently
  if (anchorType != "page") {  
    KoTextAnchor *anchor = new KoTextAnchor(shape);
    anchor->loadOffsetFromShape(element);
    d->textSharedData->shapeInserted(shape, element, d->context);

    KoTextDocumentLayout *layout
    = qobject_cast<KoTextDocumentLayout*>(cursor.block().document()->documentLayout());
    if (layout) {
      KoInlineTextObjectManager *textObjectManager = layout->inlineTextObjectManager();
      if (textObjectManager) {
        textObjectManager->insertInlineObject(cursor, anchor);
      }
    }
  }
}
}
```
Sample 10 Source Original

```c
void KisQPainterCanvas::tabletEvent(QTabletEvent *e)
{
    setFocus(Qt::OtherFocusReason);
    m_d->blockMouseEvent.start(100);
    qreal subpixelX = e->hiResGlobalX();
    subpixelX = subpixelX - ((int) subpixelX); // leave only part behind the dot
    qreal subpixelY = e->hiResGlobalY();
    subpixelY = subpixelY - ((int) subpixelY); // leave only part behind the dot
    QPointF pos(e->x() + subpixelX + m_d->documentOffset.x() - m_d->origin.x(),
                e->y() + subpixelY + m_d->documentOffset.y() - m_d->origin.y());
    m_d->toolProxy->tabletEvent(e, m_d->viewConverter->viewToDocument(pos));
}
```

Sample 10 Source Modified

```c
void KisQPainterCanvas::tabletEvent(QTabletEvent *e)
{
    setFocus(Qt::OtherFocusReason);
    m_d->blockMouseEvent.start(100);
    m_d->toolProxy->tabletEvent(e, m_d->viewConverter->viewToDocument(
        e->hiResGlobalPos() - mapToGlobal(QPoint(0,0)) + m_d->documentOffset - m_d->origin ));
}
```
Sample 10 GNU `diff`

```c
void KisQPainterCanvas::tabletEvent(QTabletEvent *e)
{
    setFocus(Qt::OtherFocusReason);
    m_d->blockMouseEvent.start(100);
    qreal subpixelX = e->hiResGlobalX();
    subpixelX = subpixelX - ((int) subpixelX); // leave only part behind the dot
    qreal subpixelY = e->hiResGlobalY();
    subpixelY = subpixelY - ((int) subpixelY); // leave only part behind the dot
    QPointF pos(e->x() + subpixelX + m_d->documentOffset.x() - m_d->origin.x(),
                 e->y() + subpixelY + m_d->documentOffset.y() - m_d->origin.y());
    m_d->toolProxy->tabletEvent(e, m_d->viewConverter->viewToDocument(pos));
    m_d->toolProxy->tabletEvent(e, m_d->viewConverter->viewToDocument(
        e->hiResGlobalPos() - mapToGlobal(QPoint(0,0)) + m_d->documentOffset - m_d->origin ));
}
```

Sample 10 `srcDiff`

```c
void KisQPainterCanvas::tabletEvent(QTabletEvent *e)
{
    setFocus(Qt::OtherFocusReason);
    m_d->blockMouseEvent.start(100);
    qreal subpixelX = e->hiResGlobalX();
    subpixelX = subpixelX - ((int) subpixelX); // leave only part behind the dot
    qreal subpixelY = e->hiResGlobalY();
    subpixelY = subpixelY - ((int) subpixelY); // leave only part behind the dot
    QPointF pos(e->x() + subpixelX + m_d->documentOffset.x() - m_d->origin.x(),
                 e->y() + subpixelY + m_d->documentOffset.y() - m_d->origin.y());
    m_d->toolProxy->tabletEvent(e, m_d->viewConverter->viewToDocument(
        e->hiResGlobalPos() - mapToGlobal(QPoint(0,0)) + m_d->documentOffset - m_d->origin
    ));
}
```
for (it1 = leads.begin(); it1 != leads.end(); ++it1)
{
    // get sheet-pointer - needed to handle inter-sheet dependencies correctly
    KSpreadSheet *sh = (*it1).sheet;
    if (!sh)
        sh = sheet;

    if (sh->dependencies()->deps->rangeDeps.contains (*it1))
    {
        QValueList<RangeDependency>::iterator it3;
        QValueList<RangeDependency> rdeps =
            sh->dependencies()->deps->rangeDeps[*it1];
        it3 = rdeps.begin();
        // erase all range dependencies of this cell in this cell-chunk
        while (it3 != rdeps.end())
            if (((*it3).cellrow == cell.row()) &&
                    ((*it3).cellcolumn == cell.column()))
                it3 = rdeps.erase (it3);
            else
                ++it3;
        // erase the list if we no longer need it
        if (rdeps.empty())
            sh->dependencies()->deps->rangeDeps.erase (*it1);
    }
for (it1 = leads.begin(); it1 != leads.end(); ++it1)
{
    //get sheet-pointer - needed to handle inter-sheet dependencies correctly
    KSpreadSheet *sh = (*it1).sheet;
    if (!sh)
        sh = sheet;

    if (sh->dependencies()->deps->rangeDeps.contains (*it1))
    {
        QValueList<RangeDependency>::iterator it3;
        it3 = sh->dependencies()->deps->rangeDeps[*it1].begin();
        //erase all range dependencies of this cell in this cell-chunk
        while (it3 != sh->dependencies()->deps->rangeDeps[*it1].end())
            if (((*it3).cellrow == cell.row()) &
                ((*it3).cellcolumn == cell.column()))
                it3 = sh->dependencies()->deps->rangeDeps[*it1].erase (it3);
            else
                ++it3;

        //erase the list if we no longer need it
        if (sh->dependencies()->deps->rangeDeps[*it1].empty())
            sh->dependencies()->deps->rangeDeps.erase (*it1);
    }
}
Sample 11 GNU `diff`

```c
for (it1 = leads.begin(); it1 != leads.end(); ++it1)
{
    // get sheet-pointer - needed to handle inter-sheet dependencies correctly
    KSpreadSheet *sh = (*it1).sheet;
    if (!sh)
        sh = sheet;

    if (sh->dependencies()->deps->rangeDeps.contains (*it1))
    {
        QValueList<RangeDependency>::iterator it3;
        - QValueList<RangeDependency> rdeps =
        -     sh->dependencies()->deps->rangeDeps[*it1];
        -     it3 = rdeps.begin();
        + it3 = sh->dependencies()->deps->rangeDeps[*it1].begin();
        // erase all range dependencies of this cell in this cell-chunk
        -     while (it3 != rdeps.end())
        +     while (it3 != sh->dependencies()->deps->rangeDeps[*it1].end())
            if ((((it3).cellrow == cell.row()) &&
                 ((it3).cellcolumn == cell.column()))
                it3 = rdeps.erase (it3);
            else
                ++it3;
        // erase the list if we no longer need it
        - if (rdeps.empty())
        + if (sh->dependencies()->deps->rangeDeps[*it1].empty())
            sh->dependencies()->deps->rangeDeps.erase (*it1);
    }
```
Sample 11 srcDiff

```c
for (it1 = leads.begin(); it1 != leads.end(); ++it1)
{
    // get sheet-pointer - needed to handle inter-sheet dependencies correctly
    KSpreadSheet *sh = (*it1).sheet;
    if (!sh)
        sh = sheet;

    if (sh->dependencies()->deps->rangeDeps.contains (*it1))
    {
        QValueList<RangeDependency>::iterator it3;
        QValueList<RangeDependency> rdeps = sh->dependencies()->deps->rangeDeps[*it1];
        it3 = sh->dependencies()->deps->rangeDeps[*it1].rdeps.begin();
        // erase all range dependencies of this cell in this cell-chunk
        while (it3 != sh->dependencies()->deps->rangeDeps[*it1].rdeps.end())
            if (((*it3).cellrow == cell.row()) &&
                ((*it3).cellcolumn == cell.column()))
                it3 = sh->dependencies()->deps->rangeDeps[*it1].rdeps.erase (it3);
            else
            ++it3;
        // erase the list if we no longer need it
        if (sh->dependencies()->deps->rangeDeps[*it1].rdeps.empty())
            sh->dependencies()->deps->rangeDeps.erase (*it1);
    }
}
```
Sample 12 Source Original

```c
while (tile = iter.tile()) {
    tile->extent().getRect(&x, &y, &width, &height);
    sprintf(str, "%d,%d,%d,%d\n", x, y, width, height);
    store->write(str, strlen(str));

    tile->lockForRead();
    store->write((char *)tile->data(), tileDataSize);
    tile->unlock();

    ++iter;
}
```

Sample 12 Source Modified

```c
while (!tile = iter.tile()) {
    tile->extent().getRect(&x, &y, &width, &height);
    sprintf(str, "%d,%d,%d,%d\n", x, y, width, height);
    store->write(str, strlen(str));

    tile->lockForRead();
    store->write((char *)tile->data(), tileDataSize);
    tile->unlock();

    ++iter;
}
```
Sample 12 GNU diff

```c
while (tile = iter.tile()) {
  tile->extent().getRect(&x, &y, &width, &height);
  sprintf(str, "%d,%d,%d,%d\n", x, y, width, height);
  store->write(str, strlen(str));

  tile->lockForRead();
  store->write((char *)tile->data(), tileDataSize);
  tile->unlock();

  ++iter;
}
```

Sample 12 srcDiff

```c
while (!tile = iter.tile()) {
  tile->extent().getRect(&x, &y, &width, &height);
  sprintf(str, "%d,%d,%d,%d\n", x, y, width, height);
  store->write(str, strlen(str));

  tile->lockForRead();
  store->write((char *)tile->data(), tileDataSize);
  tile->unlock();

  ++iter;
}
```
APPENDIX D

Source-code Changes Used in the Evaluation

This contains the source-code change example and samples used in the evaluation. For each, the original and modified code, GitHub view, Mergely, GumTree, and srcDiff are provided each as a separate image. In the actual study, the original and modified of each representation was displayed side-by-side, but as we are not able to do so for all samples, we place them one after the other. Even one after the other, some images may be altered to fit. Also of note, GumTree also contained Javascript and a legend as part of the output. These are not reproduced here. The complete survey and the approach output is available at sdml.cs.kent.edu/srcDiff/dissertation/evaluation as part of the replication package.

Example Source Original

```
    Original

    foo bar = get_foo();
    if(bar != null && !bar.empty())
       bar.first().compute();
    bar = null;
```
Example Source Modified

```
foo bar = foo();
boolean is_empty = (bar != null && !bar.empty());
if(is_empty)
    bar.first().compute();
```

Example GitHub Original

```
@@ -1,4 +1,4 @@
1    -foo bar = get_foo();
2    -if(bar != null && !bar.empty())
3    
3    bar.first().compute();
4    -bar = null;
```

Example GitHub Modified

```
+foo bar = foo();
+boolean is_empty = (bar != null && !bar.empty());
+if(is_empty)
+    bar.first().compute();
```
Example Mergely Original

```
1 | foo bar = get_foo();
2 | if(bar != null &amp;&amp; !bar.empty())
   |     bar.first().compute();
   | bar = null;
```

Example Mergely Modified

```
1 | foo bar = foo();
2 | boolean is_empty = (bar != null &amp;&amp; !bar.empty());
3 | if(is_empty)
   |     bar.first().compute();
```

Example srcDiff Original

```
Original
1 | foo bar = get_foo();
2 | if(bar != null &amp;&amp; !bar.empty())
   |     bar.first().compute();
   | bar = null;
```
Example srcDiff Modified

```
Modified

1 foo bar = foo();
2 boolean is_empty = (bar != null && !bar.empty());
3 if(is_empty)
4   bar.first().compute();
```

Example GumTree Original

```
foo bar = get_foo();
if(bar != null && !bar.empty())
  bar.first().compute();
bar = null;
```

Example GumTree Modified

```
foo bar = foo();
boolean is_empty = (bar != null && !bar.empty());
if(is_empty)
  bar.first().compute();
```
```java
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
        .startObject().startObject("type")
        .startObject("properties").startObject("point")
        .field("type", "geo_point").field("lat_lon", true)
        .field("geohash", true).endObject().endObject()
        .endObject().endObject().string();

    DocumentMapper defaultMapper
    = MapperTestUtils.newParser().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "l", XContentFactory
        .jsonBuilder()
        .startObject()
        .field("point", GeoHashUtils.encode(1.2, 1.3))
        .endObject()
        .bytes());

    assertThat(doc.rootDoc().getField("point.lat"), notNullValue());
    assertThat(doc.rootDoc().getField("point.lon"), notNullValue());
    assertThat(doc.rootDoc().get("point.geohash"),
        equalTo(GeoHashUtils.encode(1.2, 1.3)));
}
```
Sample 1 Source Modified

```java
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
        .startObject().startObject("type")
        .startObject("properties").startObject("point")
        .field("type", "geo_point").field("lat_lon", true)
        .field("geohash", true).endObject().endObject()
        .endObject().endObject().string();

    DocumentMapper defaultMapper
        = createIndex("test").mapperService().documentMapperParser().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "1", XContentFactory
        .jsonBuilder()
        .startObject()
        .field("point", GeoHashUtils.encode(1.2, 1.3))
        .endObject()
        .bytes());

    assertThat(doc.rootDoc().getField("point.lat"), notNullValue());
    assertThat(doc.rootDoc().getField("point.lon"), notNullValue());
    assertThat(doc.rootDoc().get("point.geohash"),
        equalTo(GeoHashUtils.encode(1.2, 1.3)));
}
```
```java
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
            .startObject()
            .startObject("type")
            .startObject("properties")
            .startObject("point")
            .field("type", "geo_point")
            .field("lat_lon", true)
            .field("geohash", true).endObject().endObject()
            .endObject().endObject().string();

    DocumentMapper defaultMapper
            = MapperTestUtils.newParser().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "1", XContentFactory
            .jsonBuilder()
            .startObject()
            .field("point", GeoHashUtils.encode(1.2, 1.3))
            .endObject()
            .bytes());

    assertThat(doc.rootDoc().getField("point.lat"), notNullValue());
    assertThat(doc.rootDoc().getField("point.lon"), notNullValue());
    assertThat(doc.rootDoc().get("point.geohash"),
               equalTo(GeoHashUtils.encode(1.2, 1.3)));}
```
Sample 1 GitHub Modified

```java
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
        .startObject().startObject("type")
        .startObject("properties").startObject("point")
        .field("type", "geo_point").field("lat_lon", true)
        .field("geohash", true).endObject().endObject()

        .endObject().endObject().string();

    DocumentMapper defaultMapper
    +  = createIndex("test").mapperService().documentMapperParse().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "1", XContentFactory
        .jsonBuilder()
        .startObject()
        .field("point", GeoHashUtils.encode(1.2, 1.3))
        .endObject()
        .bytes());

    assertEquals(doc.rootDoc().getField("point.lat"), notNullValue());
    assertEquals(doc.rootDoc().getField("point.lon"), notNullValue());
    assertEquals(doc.rootDoc().get("point.geohash"),
        GeoHashUtils.encode(1.2, 1.3));
}
```
Sample 1 Mergely Original

```java
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
        .startObject().startObject("type")
        .startObject("properties").startObject("point")
        .field("type", "geo_point").field("lat_lon", true)
        .field("geohash", true).endObject().endObject()
        .endObject().endObject().string();

    DocumentMapper defaultMapper = MapperTestUtils.newParser().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "1", XContentFactory
        .jsonBuilder()
        .startObject()
        .field("point", GeoHashUtil.encode(1.2, 1.3))
        .endObject()
        .bytes());

    assertThat(doc.rootDoc().getField("point.lat"), notNullValue());
    assertThat(doc.rootDoc().getField("point.lon"), notNullValue());
    assertThat(doc.rootDoc().get("point.geohash"),
        equalTo(GeoHashUtils.encode(1.2, 1.3)));
}
```
Sample 1 Mergely Modified

```java
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
        .startObject().startObject("type")
        .startObject("properties").startObject("point")
        .field("type", "geo_point").field("lat_lon", true)
        .field("geohash", true).endObject().endObject()
        .endObject().endObject().string();

    DocumentMapper defaultMapper
        = createIndex("test").mapperService().documentMapperParser().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "1", XContentFactory
        .jsonBuilder()
        .startObject()
        .field("point", GeoHashUtils.encode(1.2, 1.3))
        .endObject()
        .bytes());

    assertEquals(doc.rootDoc().getField("point.lat"), notNullValue());
    assertEquals(doc.rootDoc().getField("point.lon"), notNullValue());
    assertEquals(doc.rootDoc().get("point.geohash"),
        GeoHashUtils.encode(1.2, 1.3));
}
```
Sample 1 srcDiff Original

```java
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
        .startObject().startObject("type")
        .startObject("properties").startObject("point")
        .field("type", "geo_point").field("lat_lon", true)
        .field("geohash", true).endObject().endObject()
        .endObject().endObject().string();

    DocumentMapper defaultMapper = MapperTestUtils.newParser().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "1", XContentFactory
        .jsonBuilder()
        .startObject()
        .startObject()
        .field("point", GeoHashUtils.encode(1.2, 1.3))
        .endObject()
        .endObject()
        .bytes());

    assertThat(doc.rootDoc().getField("point.lat"), notNullValue());
    assertThat(doc.rootDoc().getField("point.lon"), notNullValue());
    assertThat(doc.rootDoc().get("point.geohash"),
                equalTo(GeoHashUtils.encode(1.2, 1.3)));
}
```
Sample 1 srcDiff Modified

```java
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
        .startObject().startObject("type")
            .startObject("properties").startObject("point")
                .field("type", "geo_point").field("lat_lon", true)
                .field("geohash", true).endObject().endObject()
            .endObject().endObject().string();

    DocumentMapper defaultMapper
        = createIndex("test").mapperService().documentMapperParser().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "1", XContentFactory
        .jsonBuilder()
        .startObject()
        .field("point", GeoHashUtils.encode(1.2, 1.3))
        .endObject()
        .bytes());

    assertThat(doc.rootDoc().getField("point.lat").notNullValue());
    assertThat(doc.rootDoc().getField("point.lon").notNullValue());
    assertThat(doc.rootDoc().get("point.geohash"),
        equalTo(GeoHashUtils.encode(1.2, 1.3)));
}
```
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
        .startObject().startObject("type")
        .startObject("properties").startObject("point")
        .field("type", "geo_point").field("lat_lon", true)
        .field("geohash", true).endObject().endObject()
        .endObject().endObject().string();

    DocumentMapper defaultMapper
        = MapperTestUtils.NEW_PARSER().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "1", XContentFactory
        .jsonBuilder()
        .startObject()
        .field("point", GeoHashUtils.encode(1.2, 1.3))
        .endObject()
        .bytes());

    assertThat(doc.rootDoc().getField("point.lat"), notNullValue());
    assertThat(doc.rootDoc().getField("point.lon"), notNullValue());
    assertThat(doc.rootDoc().get("point.geohash"),
        equalTo(GeoHashUtils.encode(1.2, 1.3)));
Sample 1 GumTree Modified

```java
@Test
public void testGeoHashValue() throws Exception {
    String mapping = XContentFactory.jsonBuilder()
        .startObject().startObject("type")
        .startObject("properties").startObject("point")
            .field("type", "geo_point").field("lat_lon", true)
            .field("geohash", true).endObject().endObject()
        .endObject().endObject().string();

    DocumentMapper defaultMapper
        = createIndex("test").mapService().DocumentMapperParser().parse(mapping);

    ParsedDocument doc = defaultMapper.parse("type", "1", XContentFactory
        .jsonBuilder()
        .startObject()
        .field("point", GeoHashUtils.encode(1.2, 1.3))
        .endObject()
        .bytes());

    assertThat(doc.rootDoc().getField("point.lat"), notNullValue());
    assertThat(doc.rootDoc().getField("point.lon"), notNullValue());
    assertThat(doc.rootDoc().get("point.geohash"),
                   equalTo(GeoHashUtils.encode(1.2, 1.3)));
```
public Field getCompletionField(ContextMapping.Context ctx,
    String input, BytesRef payload) {
    final String originalInput = input;
    if (input.length() > maxInputLength) {
        final int len = correctSubStringLen(input,
            Math.min(maxInputLength, input.length()));
        input = input.substring(0, len);
    }
    for (int i = 0; i < input.length(); i++) {
        if (isReservedChar(input.charAt(i))) {
            throw new IllegalArgumentException("Illegal input [" + originalInput + "] UTF-16 codepoint [0x" + Integer.toHexString((int) input.charAt(i)).toUpperCase(Locale.ROOT) + "] at position " + i + " is a reserved character");
        }
    }
    return new SuggestField(fieldType.names().indexName(),
            ctx, input, this.fieldType, payload,
            fieldType().analyzingSuggestLookupProvider);
}
public Field getCompletionField(ContextMapping.Context ctx,
        String input, BytesRef payload) {
    final String originalInput = input;
    if (input.length() > maxInputLength) {
        final int len = correctSubStringLen(input,
            Math.min(maxInputLength, input.length()));
        input = input.substring(0, len);
    }
    for (int i = 0; i < input.length(); i++) {
        if (isReservedChar(input.charAt(i))) {
            throw new IllegalArgumentException("Illegal input "
                + originalInput + "] UTF-16 codepoint [0x" +
                Integer.toHexString((int) input.charAt(i))
                .toUpperCase(Locale.ROOT)
                + "] at position " + i + " is a reserved character");
        }
    }
    return new SuggestField(fieldType().names().indexName(),
        ctx, input, fieldType(), payload,
        fieldType().analyzingSuggestLookupProvider);
}
```java
... @@ -1,21 +1,21 @@
 1 public Field getCompletionField(ContextMapping.Context ctx,
 2     String input, BytesRef payload) {
 3     final String originalInput = input;
 4     if (input.length() > maxInputLength) {
 5         final int len = correctSubStringLen(input,
 6             Math.min(maxInputLength, input.length()));
 7         input = input.substring(0, len);
 8     }
 9     for (int i = 0; i < input.length(); i++) {
10         if (isReservedChar(input.charAt(i))) {
11             throw new IllegalArgumentException("Illegal input "
12                 + originalInput + "] UTF-16 codepoint [0x" +
13                 Integer.toHexString((int) input.charAt(i)).
14                 toUpperCase(Locale.ROOT) + "] at position " + i + " is a reserved character");
15         }
16     }
17
18     return new SuggestField(fieldType.names().indexName(),
19         ctx, input, this.fieldType, payload,
20         fieldType().analyzingSuggestLookupProvider);
21 }
```
```java
public Field getCompletionField(ContextMapping.Context ctx,
    String input, BytesRef payload) {
    final String originalInput = input;
    if (input.length() > maxInputLength) {
        final int len = correctSubStringLen(input,
            Math.min(maxInputLength, input.length()));
        input = input.substring(0, len);
    }
    for (int i = 0; i < input.length(); i++) {
        if (isReservedChar(input.charAt(i))) {
            throw new IllegalArgumentException("Illegal input [
                originalInput + "] UTF-16 codepoint [0x" + Integer.toHexString((int) input.charAt(i))
                .toUpperCase(Locale.ROOT) + "] at position " + i + " is a reserved character");
        }
    }
    return new SuggestField(fieldType().names().indexName(),
        ctx, input, fieldType(), payload,
        fieldType().analyzingSuggestLookupProvider);
}
```
public Field getCompletionField(ContextMapping.Context ctx,
        String input, BytesRef payload) {
    final String originalInput = input;
    if (input.length() > maxInputLength) {
        final int len = correctSubStringLen(input,
                Math.min(maxInputLength, input.length()));
        input = input.substring(0, len);
    }
    for (int i = 0; i < input.length(); i++) {
        if (isReservedChar(input.charAt(i))) {
            throw new IllegalArgumentException("Illegal input [" +
                    originalInput + "] UTF-16 codepoint " +
                    Integer.toHexString((int) input.charAt(i)) +
                    .toUpperCase(Locale.ROOT) + "] at position " + i + " is a reserved character");
        }
    }
    return new SuggestField(this.fieldType().names().indexName(),
            ctx, input, this.fieldType(), payload,
            fieldType().analyzingSuggestLookupProvider);
}
public Field getCompletionField(ContextMapping.Context ctx,  
    String input, BytesRef payload) {
  final String originalInput = input;
  if (input.length() > maxInputLength) {
    final int len = correctSubStringLen(input,  
      Math.min(maxInputLength, input.length()));  
    input = input.substring(0, len);  
  }
  for (int i = 0; i < input.length(); i++) {
    if (isReservedChar(input.charAt(i))) {
      throw new IllegalArgumentException("Illegal input ["  
        + originalInput + "] UTF-16 codepoint [0x"  
        + Integer.toHexString((int) input.charAt(i))  
        .toUpperCase(Locale.ROOT)  
        + "] at position " + i + " is a reserved character");
    }
  }
  return new SuggestField(fieldType().names().indexName().  
    .ctx, input, fieldType(), payload,  
    fieldType().analyzingSuggestLookupProvider);
Sample 2 srcDiff Original

```java
public Field getCompletionField(ContextMapping.Context ctx,
        String input, BytesRef payload) {
    final String originalInput = input;
    if (input.length() > maxInputLength) {
        final int len = correctSubStringLen(input,
                Math.min(maxInputLength, input.length()));
        input = input.substring(0, len);
    }
    for (int i = 0; i < input.length(); i++) {
        if (isReservedChar(input.charAt(i))) {
            throw new IllegalArgumentException("Illegal input ",
                    + originalInput + "] UTF-16 codepoint [0x" +
                    + Integer.toHexString((int) input.charAt(i))
                    + ".toUpperCase(Locale.ROOT)
                    + "] at position " + i + " is a reserved character");
        }
    }
    return new SuggestField(fieldType().names().indexName(),
            ctx, input, this.fieldType, payload,
            fieldType().analyzingSuggestLookupProvider);
}
```
Sample 2 srcDiff Modified

```java
public Field getCompletionField(ContextMapping.Context ctx,
        String input, BytesRef payload) {
    final String originalInput = input;
    if (input.length() > maxInputLength) {
        final int len = correctSubStringLen(input,
            Math.min(maxInputLength, input.length()));
        input = input.substring(0, len);
    }
    for (int i = 0; i < input.length(); i++) {
        if (isReservedChar(input.charAt(i))) {
            throw new IllegalArgumentException("Illegal input [
                + originalInput + " UTF-16 codepoint [0x" +
            Integer.toHexString((int) input.charAt(i))
                + "] at position " + i + " is a reserved character");
        }
    }
    return new SuggestField(fieldType().names().indexName(),
        ctx, input, fieldType(), payload,
        fieldType().analysingSuggestLookupProvider);
}
```
public Field getCompletionField(ContextMapping.Context ctx,  
        String input, BytesRef payload) {  
    final String originalInput = input;  
    if (input.length() > maxInputLength) {  
        final int len = correctSubStringLen(input,  
                Math.min(maxInputLength, input.length()));  
        input = input.substring(0, len);  
    }  
    for (int i = 0; i < input.length(); i++) {  
        if (isReservedChar(input.charAt(i))) {  
            throw new IllegalArgumentException("Illegal input ["  
                    + originalInput + "] UTF-16 codepoint [0x"  
                    + Integer.toHexString((int) input.charAt(i))  
                    + "] at position " + i + " is a reserved character");  
        }  
    }  
    return new SuggestField("fieldType".names().indexName(),  
        ctx, input, 
        this.fieldType, payload,  
        fieldType().analyzingSuggestLookupProvider());  
}
```java
public Field getCompletionField(ContextMapping.Context ctx,
        String input, BytesRef payload) {
    final String originalInput = input;
    if (input.length() > maxInputLength) {
        final int len = correctSubStringLen(input,
                Math.min(maxInputLength, input.length()));
        input = input.substring(0, len);
    }
    for (int i = 0; i < input.length(); i++) {
        if (isReservedChar(input.charAt(i))) {
            throw new IllegalArgumentException("Illegal input [
                    + originalInput + "] UTF-16 codepoint [0x"
                    + Integer.toHexString((int) input.charAt(i))
                    .toUpperCase(Locale.ROOT)
                    + "] at position " + i + " is a reserved character");
        }
    }
    return new SuggestField(fieldType().names().indexName(),
            ctx, input, fieldType(), payload,
            fieldType().analyzingSuggestLookupProvider);
}
```
```java
@Test
public void passQueryAsJSONStringTest() throws Exception {
    client().admin().indices().prepareCreate("test").setSettings(
        ImmutableSettings.settingsBuilder().put("index.number_of_shards", 1)).execute().actionGet();

    client().prepareIndex("test", "type1", "1")
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).execute().actionGet();

    WrapperQueryBuilder wrapper
        = new WrapperQueryBuilder("{ \"term\" : { \"field1\" : \"value1_1\" } }");
    CountResponse countResponse = client().prepareCount()
        .setQuery(wrapper).execute().actionGet();
    assertHitCount(countResponse, 1l);

    BoolQueryBuilder bool = new BoolQueryBuilder();
    bool.must(wrapper);
    bool.must(new TermQueryBuilder("field2", "value2_1"));

    countResponse = client().prepareCount().setQuery(wrapper).execute().actionGet();
    assertHitCount(countResponse, 1l);
}
```
Sample 3 Source Modified

```java
@Test
public void passQueryAsJSONObjectTest() throws Exception {
    assertAcked(prepareCreate("test").setSettings(SETTING_NUMBER_OF_SHARDS, 1));

    client().prepareIndex("test", "type1", "1")
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).get();

    WrapperQueryBuilder wrapper
        = new WrapperQueryBuilder("{ \"term\" : { \"field1\" : \"value1_1\" } }");
    assertHitCount(client().prepareCount()
        .setQuery(wrapper).get(), 11);

    BoolQueryBuilder bool = boolQuery().must(wrapper).must(new TermQueryBuilder( "field2", "value2_1"));
    assertHitCount(client().prepareCount().setQuery(bool).get(), 11);
}
```
Sample 3 GitHub Original

```java
@Text
public void passQueryAsJSONStringTest() throws Exception {
    client().admin().indices().prepareCreate("test").setSettings( 
        ImmutableSettings.settingsBuilder().put("index.number_of_shards", 1)).execute().actionGet();
    client().prepareIndex("test", "type1", "1")
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).execute().actionGet();

    WrappingQueryBuilder wrapper
        = new WrappingQueryBuilder("{ \"term\" : { \"field1\" : \"value1_1\" } }" );
    CountResponse countResponse = client().prepareCount()
        .setQuery(wrapper).execute().actionGet();
    assertHitCount(countResponse, 1);

    BoolQueryBuilder bool = new BoolQueryBuilder();
    bool.must(wrapper);
    bool.must(new TermQueryBuilder("field2", "value2_1"));
    countResponse = client().prepareCount().setQuery(wrapper).execute().actionGet();
    assertHitCount(countResponse, 1);
}
```
Sample 3 GitHub Modified

```java
@Test
public void passQueryAsJSONStringTest() throws Exception {
    assertAcked(prepareCreate("test").setSettings(SETTING_NUMBER_OF_SHARDS,
        1));

    client().prepareIndex("test", "type1", "1"
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).get();

    WrapperQueryBuilder wrapper
        = new WrapperQueryBuilder("{ \"term\" : { \"field1\" : \"value1_1\" } }");

    assertHitCount(client().prepareCount()
        .setQuery(wrapper).get(), 11);

    BoolQueryBuilder bool = boolQuery().must(wrapper).must(new TermQueryBuilder("field2", "value2_1"));

    assertHitCount(client().prepareCount().setQuery(bool).get(), 11);
}
```
Sample 3 Mergely Original

```java
@Test
public void passQueryAsJSONStringTest() throws Exception {
    client().admin().indices().prepareCreate("test").setSettings(  
        ImmutableSettings.settingsBuilder().put("index.number_of_shards", 1)).execute().actionGet();

    client().prepareIndex("test", "type1", "1")
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).execute().actionGet();

    WrapperQueryBuilder wrapper
        = new WrapperQueryBuilder("{ "term" : { "field1" : "value1_1" } }");

    CountResponse countResponse = client().prepareCount()
        .setQuery(wrapper).execute().actionGet();

    assertHitCount(countResponse, 11);

    BoolQueryBuilder bool = new BoolQueryBuilder();
    -bool.must(wrapper);
    -bool.must(new TermQueryBuilder("field2", "value2_1"));

    countResponse = client().prepareCount().setQuery(wrapper).execute().actionGet();
    assertHitCount(countResponse, 11);
}
```

Sample 3 Mergely Modified

```java
@Test
public void passQueryAsJSONStringTest() throws Exception {

    assertAcked(prepareCreate("test").setSettings(SETTING_NUMBER_OF_SHARDS, 1));

    client().prepareIndex("test", "type1", "1")
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).get();

    WrapperQueryBuilder wrapper
        = new WrapperQueryBuilder("{ "term" : { "field1" : "value1_1" } }");

    assertHitCount(client().prepareCount()
        .setQuery(wrapper).get(), 11);

    BoolQueryBuilder bool = boolQuery().must(wrapper).must(new TermQueryBuilder("field2", "value2_1"));
    assertHitCount(client().prepareCount().setQuery(bool).get(), 11);
}
```
Sample 3 srcDiff Original

```java
@Test
public void passQueryAsJSONStringTest() throws Exception {
    client().admin().indices().prepareCreate("test").setSettings(
            ImmutableSettings.settingsBuilder().put("index.number_of_shards", 1)).execute().actionGet();

    client().prepareIndex("test", "type1", "1")
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).execute().actionGet();

    WrapperQueryBuilder wrapper
        = new WrapperQueryBuilder("{ "term" : { "field1" : "value1_1" } }")
        .setQuery(wrapper).execute().actionGet();

    BooleanQuery booleanQuery = new BooleanQuery();
    booleanQuery.must(new TermQueryBuilder("field2", "value2_1"));

    booleanQuery = new BooleanQuery();
    booleanQuery.must(new TermQueryBuilder("field2", "value2_1"));

    CountResponse countResponse = client().prepareCount().setQuery(boolQuery).execute().actionGet();
    assertHitCount(countResponse, 1);
}
```

Sample 3 srcDiff Modified

```java
@Test
public void passQueryAsJSONStringTest() throws Exception {
    assertAcked(prepareCreate("test").setSettings(SETTING_NUMBER_OF_SHARDS, 1));

    client().prepareIndex("test", "type1", "1")
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).get();

    WrapperQueryBuilder wrapper
        = new WrapperQueryBuilder("{ "term" : { "field1" : "value1_1" } }");

    assertHitCount(client().prepareCount().setQuery(wrapper).get(), 1);

    BooleanQuery booleanQuery = BooleanQueryBuilders.must(new TermQueryBuilder("field2", "value2_1"));

    assertHitCount(client().prepareCount().setQuery(booleanQuery).get(), 1);
}
```
Sample 3 GumTree Original

```java
@Test
public void passQueryAsJSONObjectTest() throws Exception {
    Client().admin().indices().prepareCreate("test").setSettings(
        ImmutableSettings.builder().put("index.number_of_shards", 1)).execute().actionGet();

    Client().prepareIndex("test", "type1", "1")
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).execute().actionGet();

    WrapperQueryBuilder wrapper
        = new WrapperQueryBuilder("{ "term" : { "field1" : "value1_1" } }");
    CountResponse countResponse = client().prepareCount()
        .setQuery(wrapper).execute().actionGet();
    assertHitCount(countResponse, 1L);

    BoolQueryBuilder bool = new BoolQueryBuilder();
    bool.must(new TermQueryBuilder("field2", "value2_1"));
    CountResponse countResponse = client().prepareCount().setQuery(wrapper).execute().actionGet();
    assertHitCount(countResponse, 1L);
}
```

Sample 3 GumTree Modified

```java
@Test
public void passQueryAsJSONObjectTest() throws Exception {
    assertAcked(prepareCreate("test").setSettings(Settings.builder().put("index.number_of_shards", 1)));

    Client().prepareIndex("test", "type1", "1")
        .setSource("field1", "value1_1", "field2", "value2_1")
        .setRefresh(true).get();

    WrapperQueryBuilder wrapper
        = new WrapperQueryBuilder("{ "term" : { "field1" : "value1_1" } }");
    assertHitCount(client().prepareCount()
        .setQuery(wrapper).get(), 1L);

    BoolQueryBuilder bool = BoolQuery().must(wrapper).must(new TermQueryBuilder("field2", "value2_1"));
    assertHitCount(client().prepareCount().setQuery(bool).get(), 1L);
}
```
Sample 4 Source Original

<table>
<thead>
<tr>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean isShutdown() { return shutdown; }</td>
</tr>
</tbody>
</table>

Sample 4 Source Modified

<table>
<thead>
<tr>
<th>Modified</th>
</tr>
</thead>
</table>
| @Override
| public boolean isShutdown() {
| return false;
| } |

Sample 4 GitHub Original

```
... @@ -1 +1,4 @@
1 -public boolean isShutdown() { return shutdown; }
```

Sample 4 GitHub Modified

```
... @@ -1 +1,4 @@
1 +@Override
2 +public boolean isShutdown() {
3 + return false;
4 +}
```
Sample 4 Mergely Original

```java
public boolean isShutdown() { return shutdown; }
```

Sample 4 Mergely Modified

```java
@override
public boolean isShutdown() {
    return false;
}
```

Sample 4 srcDiff Original

```
Original

public boolean isShutdown() {
    return shutdown;
}
```
Sample 4 srcDiff Modified

```java
@Override
public boolean isShutdown() {
    return false;
}
```

Sample 4 GumTree Original

```java
public boolean isShutdown() { return shutdown; }
```

Sample 4 GumTree Modified

```java
@Override
public boolean isShutdown() {
    return false;
}
```
Sample 5 Source Original

```java
@Override public boolean containsAll(Collection<? super T> targets) {
    try {
        return range.containsAll((Iterable<? extends C>) targets);
    } catch (ClassCastException e) {
        return false;
    }
}
```

Sample 5 Source Modified

```java
@Override public boolean containsAll(Collection<? super T> targets) {
    return Collections2.containsAllImpl(this, targets);
}
```

Sample 5 GitHub Original

```diff
- @Override public boolean containsAll(Collection<? super T> targets) {
-     try {
-         return range.containsAll((Iterable<? extends C>) targets);
-     } catch (ClassCastException e) {
-         return false;
-     }
- }
+ @Override public boolean containsAll(Collection<? super T> targets) {
+     return Collections2.containsAllImpl(this, targets);
+ }
```
Sample 5 GitHub Modified

```java
@Override public boolean containsAll(Collection<?> targets) {
    return Collections2.containsAllImpl(this, targets);
}
```

Sample 5 Mergely Original

```java
@Override public boolean containsAll(Collection<?> targets) {
    try {
        return range.containsAll((Iterable<? extends C>) targets);
    } catch (ClassCastException e) {
        return false;
    }
}
```

Sample 5 Mergely Modified

```java
@Override public boolean containsAll(Collection<?> targets) {
    return Collections2.containsAllImpl(this, targets);
}
```
### Sample 5 srcDiff Original

```java
@override public boolean containsAll(Collection<?> targets) {
    try {
        return range.containsAll((Iterable<? extends C>) targets);
    } catch (ClassCastException e) {
        return false;
    }
}
```

### Sample 5 srcDiff Modified

```java
@override public boolean containsAll(Collection<?> targets) {
    return Collections2.containsAllImpl(this, targets);
}
```

### Sample 5 GumTree Original

```java
@Override public boolean containsAll(Collection<?> targets) {
    try {
        return range.containsAll((Iterable<? extends C>) targets);
    } catch (ClassCastException e) {
        return false;
    }
}
```
Sample 5 GumTree Modified

```java
@Override
public boolean containsAll(Collection<?> targets) {
    return Collections2.containsAllImpl(this, targets);
}
```
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null;
            e = e.getNext()) {
            if (e.getHash() != hash) {
                continue;
            }

            K entryKey = e.getKey();
            if (entryKey == null) {
                continue;
            }

            if (keyEquivalence.equivalent(key, entryKey)) {
                if (expires() && isExpired(e)) {
                    return null;
                }
                if (isInvalid(e)) {
                    return null;
                }

                recordRead(e);
                return e;
            }
        }
    }

    return null;
}
```java
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null; 
            e = e.getNext()) {
            if (e.getHash() != hash) {
                continue;
            }

        K entryKey = e.getKey();
        if (entryKey == null) {
            continue;
        }

        if (keyEquivalence.equivalent(key, entryKey)) {
            if (isLive(e)) {
                recordRead(e);
                return e;
            }

            return null;
        }
    }
    return null;
}
```
Sample 6 GitHub Original

```java
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null; 
            e = e.getNext()) {
            if (e.getHash() != hash) {
                continue;
            }
        
    K entryKey = e.getKey();
    if (entryKey == null) {
        continue;
    }
    
    if (keyEquivalence.equivalent(key, entryKey)) {
        - if (expires() && isExpired(e)) {
        -     return null;
        - }
        - if (isValid(e)) {
        -     return null;
        - }
        
        recordRead(e);
        return e;
    }

    }
}
return null;
```
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null; 
             e = e.getNext()) {
            if (e.getHash() != hash) { 
                continue;
            }
        
        K entryKey = e.getKey();
        if (entryKey == null) {
            continue;
        }
        
        if (keyEquivalence.equivalent(key, entryKey)) {
            if (isLive(e)) {

                recordRead(e);

                return e;
            
        +        return null;
            
            +    }
        
        
            +}
        
    
    return null;
}
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null;
            e = e.getNext()) {
            if (e.getHash() != hash) {
                continue;
            }

            K entryKey = e.getKey();
            if (entryKey == null) {
                continue;
            }

            if (keyEquivalence.equivalent(key, entryKey)) {
                if (!expires() && !isExpired(e)) {
                    return null;
                }

                if (isInvalid(e)) {
                    return null;
                }

                recordRead(e);
                return e;
            }
        }
    }

    return null;
}
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null;
            e = e.getNext()) {
            if (e.getHash() != hash) {
                continue;
            }

            K entryKey = e.getKey();
            if (entryKey == null) {
                continue;
            }

            if (keyEquivalence.equivalent(key, entryKey)) {
                if (isLive(e)) {
                    recordRead(e);
                    return e;
                }
            }
        }
    }
    return null;
}
Sample 6 srcDiff Original

```java
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null;
            e = e.getNext()) {
            if (e.getHash() != hash) {
                continue;
            }

            K entryKey = e.getKey();
            if (entryKey == null) {
                continue;
            }

            if (keyEquivalence.equivalent(key, entryKey)) {
                if (expires() && isExpired(e)) {
                    return null;
                }
                if (isInvalid(e)) {
                    return null;
                }

            recordRead(e);

            return e;
        }
    }
    return null;
}
```
```java
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null;
            e = e.getNext()) {
            if (e.getHash() != hash) {
                continue;
            }

            K entryKey = e.getKey();
            if (entryKey == null) {
                continue;
            }

            if (keyEquivalence.equivalent(key, entryKey)) {

                if (isLive(e)) {
                    recordRead(e);
                    return e;
                }

                return null;
            }
        }
    }

    return null;
}
```
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null; 
            e = e.getNext()) {
            if (e.getHash() != hash) {
                continue;
            }
        }
        K entryKey = e.getKey();
        if (entryKey == null) {
            continue;
        }
        if (keyEquivalence.equivalent(key, entryKey)) {
            if (expires() && Expired[e]) {
                return null;
            }
            if (isInvalid(e)) {
                return null;
            }
        }
    }
    recordRead(e);
    return e;
}

return null;
Sample 6 GumTree Modified

```java
public ReferenceEntry<K, V> getLiveEntry(Object key, int hash) {
    if (count != 0) {
        for (ReferenceEntry<K, V> e = getFirst(hash); e != null; 
            e = e getNext()) {
            if (e.getHash() != hash) {
                continue;
            }
        }

        K entryKey = e.getKey();
        if (entryKey == null) {
            continue;
        }

        if (keyEquivalence.equivalent(key, entryKey)) {
            if ([isLive[e]] { 
                recordRead(e);
                return e;
            }

            return null;
        }
    }

    return null;
}
```
Sample 7 Source Original

```
private void convertFile(File file) {
    try {
        InplaceFileConverter fc = new InplaceFileConverter(lineConverter);
        byte[] ba = fc.readFile(file);
        fc.convert(file, ba);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```

Sample 7 Source Modified

```
private void convertFile(File file) {
    try {
        InplaceFileConverter fc = new InplaceFileConverter(ruleSet);
        fc.convert(file);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```
Sample 7 GitHub Original

```java
private void convertFile(File file) {
    try {
        - InplaceFileConverter fc = new InplaceFileConverter(lineConverter);
        - byte[] ba = fc.readFile(file);
        - fc.convert(file, ba);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```

Sample 7 GitHub Modified

```java
private void convertFile(File file) {
    try {
        + InplaceFileConverter fc = new InplaceFileConverter(ruleSet);
        + fc.convert(file);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```
Sample 7 Mergely Original

```java
private void convertFile(File file) {
    try {
        InplaceFileConverter fc = new InplaceFileConverter(lineConverter);
        byte[] ba = fc.readFile(file);
        fc.convert(file, ba);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```

Sample 7 Mergely Modified

```java
private void convertFile(File file) {
    try {
        InplaceFileConverter fc = new InplaceFileConverter(ruleSet);
        fc.convert(file);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```
Sample 7 srcDiff Original

```java
private void convertFile(File file) {
    try {
        InplaceFileConverter fc = new InplaceFileConverter(lineConverter);
        byte[] ba = fc.readFile(file);
        fc.convert(file, ba);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```

Sample 7 srcDiff Modified

```java
private void convertFile(File file) {
    try {
        InplaceFileConverter fc = new InplaceFileConverter(ruleSet);
        fc.convert(file);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```
Sample 7 GumTree Original

```java
private void convertFile(File file) {
    try {
        InplaceFileConverter fc = new InplaceFileConverter(incConverter);
        byte[] ba = fc.readFile(file);
        fc.convert(file, ba);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```

Sample 7 GumTree Modified

```java
private void convertFile(File file) {
    try {
        InplaceFileConverter fc = new InplaceFileConverter(ruleset);
        fc.convert(file);
    } catch (IOException exc) {
        addException(new ConversionException(exc.toString()));
    }
}
```
Sample 8 Source Original

```java
public static void assertDuration(double currentDuration,
       long referenceDuration, double referenceBIPS)
    throws AssertionFailedError {
    double adjustedDuration = adjustExpectedDuration(referenceDuration,
        referenceBIPS);
    if (currentDuration > adjustedDuration * SLACK_FACTOR) {
        throw new AssertionFailedError(currentDuration
            + " exceeded expected "
            + adjustedDuration + " (adjusted), "
            + referenceDuration + " (raw)";
    }
}
```

Sample 8 Source Modified

```java
public static void assertDuration(double currentDuration,
       long referenceDuration, double referenceBIPS)
    throws AssertionFailedError {
    double adjustedDuration = adjustExpectedDuration(referenceDuration,
        referenceBIPS);
    if (currentDuration > adjustedDuration * SLACK_FACTOR) {
        throw new AssertionFailedError("current duration " + currentDuration
            + " exceeded expected "
            + adjustedDuration + " (adjusted reference), "
            + referenceDuration + " (raw reference)";
    }
```
Sample 8 GitHub Original

```java
public static void assertDuration(double currentDuration,
    long referenceDuration, double referenceBIPS)
    throws AssertionError {
    double adjustedDuration = adjustExpectedDuration(referenceDuration,
        referenceBIPS);
    if (currentDuration > adjustedDuration * SLACK_FACTOR) {
        throw new AssertionError("current duration exceeded expected "
            + " adjusted duration " + " (adjusted), "
            + " referenceDuration " + " (raw)");
    }
}
```

Sample 8 GitHub Modified

```java
public static void assertDuration(double currentDuration,
    long referenceDuration, double referenceBIPS)
    throws AssertionError {
    double adjustedDuration = adjustExpectedDuration(referenceDuration,
        referenceBIPS);
    if (currentDuration > adjustedDuration * SLACK_FACTOR) {
        throw new AssertionError("current duration exceeded expected "
            + " adjusted duration " + " (adjusted reference), "
            + " referenceDuration " + " (raw reference)");
    }
}
```
Sample 8 Mergely Original

```java
public static void assertDuration(double currentDuration,
        long referenceDuration, double referenceBIPS)
    throws AssertionError {
    double adjustedDuration = adjustExpectedDuration(referenceDuration,
            referenceBIPS);
    if (currentDuration > adjustedDuration * SLACK_FACTOR) {
        throw new AssertionError("current_duration " + currentDuration
            + " exceeded expected "
            + adjustedDuration + " (adjusted reference)",
            + referenceDuration + " (raw reference)");
    }
}
```

Sample 8 Mergely Modified

```java
public static void assertDuration(double currentDuration,
        long referenceDuration, double referenceBIPS)
    throws AssertionError {
    double adjustedDuration = adjustExpectedDuration(referenceDuration,
            referenceBIPS);
    if (currentDuration > adjustedDuration * SLACK_FACTOR) {
        throw new AssertionError("current_duration " + currentDuration
            + " exceeded expected "
            + adjustedDuration + " (adjusted reference)",
            + referenceDuration + " (raw reference)");
    }
}
```
Sample 9 srcDiff Original

```java
public static void assertDuration(double currentDuration,
                                  long referenceDuration, double referenceBIPS)
    throws AssertionFailedError {
    double adjustedDuration = adjustExpectedDuration(referenceDuration,
                                                     referenceBIPS);
    if (currentDuration > adjustedDuration * SLACK_FACTOR) {
        throw new AssertionFailedError("current duration +
                                         " + exceeded expected "
                                         + adjustedDuration + " (adjusted), "
                                         + referenceDuration + " (raw)"");
    }
}
```

Sample 8 srcDiff Modified

```java
public static void assertDuration(double currentDuration,
                                  long referenceDuration, double referenceBIPS)
    throws AssertionFailedError {
    double adjustedDuration = adjustExpectedDuration(referenceDuration,
                                                     referenceBIPS);
    if (currentDuration > adjustedDuration * SLACK_FACTOR) {
        throw new AssertionFailedError("current duration " + currentDuration
                                         + " exceeded expected "
                                         + adjustedDuration + " (adjusted reference), "
                                         + referenceDuration + " (raw reference)"");
    }
}
```
Sample 8 GumTree Original

```java
public static void assertDuration(double currentDuration,
   long referenceDuration, double referenceBIPS)
   throws AssertionFailedError {
   double adjustedDuration = adjustExpectedDuration(referenceDuration,
      referenceBIPS);
   if (currentDuration > adjustedDuration * SLACK_FACTOR) {
      throw new AssertionFailedError(currentDuration
         + " exceeded expected "
         + adjustedDuration + " (adjusted), "
         + referenceDuration + " (raw)"};
   }
}
```

Sample 8 GumTree Modified

```java
public static void assertDuration(double currentDuration,
   long referenceDuration, double referenceBIPS)
   throws AssertionFailedError {
   double adjustedDuration = adjustExpectedDuration(referenceDuration,
      referenceBIPS);
   if (currentDuration > adjustedDuration * SLACK_FACTOR) {
      throw new AssertionFailedError("current duration " + currentDuration
         + " exceeded expected "
         + adjustedDuration + " (adjusted reference), "
         + referenceDuration + " (raw reference)"};
   }
}
```
Sample 9 Source Original

```
public synchronized boolean remove(Marker markerToRemove) {
    if (children == null) {
        return false;
    }

    int size = children.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) children.get(i);
        if (markerToRemove.equals(m)) {
            children.remove(i);
            return true;
        }
    }

    return false;
}
```

Sample 9 Source Modified

```
public synchronized boolean remove(Marker referenceToRemove) {
    if (referenceList == null) {
        return false;
    }

    int size = referenceList.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) referenceList.get(i);
        if (referenceToRemove.equals(m)) {
            referenceList.remove(i);
            return true;
        }
    }

    return false;
}
```
Sample 9 GitHub Original

```java
public synchronized boolean remove(Marker markerToRemove) {
    if (children == null) {
        return false;
    }

    int size = children.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) children.get(i);
        if (markerToRemove.equals(m)) {
            children.remove(i);
            return true;
        }
    }

    return false;
}
```
Sample 9 GitHub Modified

```java
public synchronized boolean remove(Marker referenceToRemove) {
    if (referenceList == null) {
        return false;
    }

    int size = referenceList.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) referenceList.get(i);
        if (referenceToRemove.equals(m)) {
            referenceList.remove(i);
            return true;
        }
    }

    return false;
}
```
Sample 9 Mergely Original

```java
public synchronized boolean remove(Marker markerToRemove) {
    if (children == null) {
        return false;
    }

    int size = children.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) children.get(i);
        if (markerToRemove.equals(m)) {
            children.remove(i);
            return true;
        }
    }

    return false;
}
```
```java
public synchronized boolean remove(Marker referenceToRemove) {
    if (referenceList == null) {
        return false;
    }

    int size = referenceList.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) referenceList.get(i);
        if (referenceToRemove.equals(m)) {
            referenceList.remove(i);
            return true;
        }
    }
    return false;
}
```
public synchronized boolean remove(Marker markerToRemove) {
    if (children == null) {
        return false;
    }

    int size = children.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) children.get(i);
        if (markerToRemove.equals(m)) {
            children.remove(i);
            return true;
        }
    }

    return false;
}
public synchronized boolean remove(Marker referenceToRemove) {
    if (referenceList == null) {
        return false;
    }
    int size = referenceList.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) referenceList.get(i);
        if (referenceToRemove.equals(m)) {
            referenceList.remove(i);
            return true;
        }
    }
    return false;
}
Sample 9 GumTree Original

```java
public synchronized boolean remove(Marker markerToRemove) {
    if (children == null) {
        return false;
    }

    int size = children.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) children.get(i);
        if (markerToRemove.equals(m)) {
            children.remove(i);
            return true;
        }
    }

    return false;
}
```

Sample 9 GumTree Modified

```java
public synchronized boolean remove(Marker referenceToRemove) {
    if (refereeList == null) {
        return false;
    }

    int size = refereeList.size();
    for (int i = 0; i < size; i++) {
        Marker m = (Marker) refereeList.get(i);
        if (referenceToRemove.equals(m)) {
            refereeList.remove(i);
            return true;
        }
    }

    return false;
}
```
static class ExpirationSpec {
    @Nullable
    private final Long expireAfterAccessMillis;
    @Nullable
    private final Long expireAfterWriteMillis;
    
    private ExpirationSpec(Long expireAfterAccessMillis, Long expireAfterWriteMillis) {
        Preconditions.checkNotNull(
            expireAfterAccessMillis == null || expireAfterWriteMillis == null);
        this.expireAfterAccessMillis = expireAfterAccessMillis;
        this.expireAfterWriteMillis = expireAfterWriteMillis;
    }
    
    public static ExpirationSpec afterAccess(long afterAccess, TimeUnit unit) {
        return new ExpirationSpec(unit.toMillis(afterAccess), null);
    }
    
    public static ExpirationSpec afterWrite(long afterWrite, TimeUnit unit) {
        return new ExpirationSpec(null, unit.toMillis(afterWrite));
    }
    
    @Override
    public int hashCode() {
        return Objects.hash(expireAfterAccessMillis, expireAfterWriteMillis);
    }
    
    @Override
    public boolean equals(Object o) {
        if (o instanceof ExpirationSpec) {
            ExpirationSpec that = (ExpirationSpec) o;
            return Objects.equals(this.expireAfterAccessMillis, that.expireAfterAccessMillis) &&
                  Objects.equals(this.expireAfterWriteMillis, that.expireAfterWriteMillis);
        }
        return false;
    }
    
    @Override
    public String toString() {
        return Objects.toStringHelper(this)
            .add("expireAfterAccessMillis", expireAfterAccessMillis)
            .add("expireAfterWriteMillis", expireAfterWriteMillis)
            .toString();
    }
}
```java
static class DurationSpec {
    private final long duration;
    private final TimeUnit unit;

    private DurationSpec(long duration, TimeUnit unit) {
        this.duration = duration;
        this.unit = unit;
    }

    public static DurationSpec of(long duration, TimeUnit unit) {
        return new DurationSpec(duration, unit);
    }

    @Override
    public int hashCode() {
        return Objects.hashCode(duration, unit);
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof DurationSpec) {
            DurationSpec that = (DurationSpec) o;
            return unit.toNanos(duration) == that.unit.toNanos(that.duration);
        }
        return false;
    }

    @Override
    public String toString() {
        return Objects.toStringHelper(this)
            .add("duration", duration)
            .add("unit", unit)
            .toString();
    }
}
```
```java
    @-1,35 +1,26 @
    - static class ExpirationSpec {
            - @Nullable
            - private final Long expireAfterAccessMillis;
            - @Nullable
            - private final Long expireAfterWriteMillis;

            - private ExpirationSpec(Long expireAfterAccessMillis, Long expireAfterWriteMillis) {
                    - Preconditions.checkNotNull(
                            - expireAfterAccessMillis == null || expireAfterWriteMillis == null);
                    - this.expireAfterAccessMillis = expireAfterAccessMillis;
                    - this.expireAfterWriteMillis = expireAfterWriteMillis;
            }

            - public static ExpirationSpec afterAccess(Long afterAccess, TimeUnit unit) {
                    - return new ExpirationSpec(unit.toMillis(afterAccess), null);
                    - }

            - public static ExpirationSpec afterWrite(Long afterWrite, TimeUnit unit) {
                    - return new ExpirationSpec(null, unit.toMillis(afterWrite));
            }

            @Override
            public int hashCode() {
                    - return Objects.hashCode(expireAfterAccessMillis, expireAfterWriteMillis);
            }

            @Override
            public boolean equals(Object o) {
                    - if (o instanceof ExpirationSpec) {
                            - ExpirationSpec that = (ExpirationSpec) o;
                            - return Objects.equals(this.expireAfterAccessMillis, that.expireAfterAccessMillis)
                            - && Objects.equals(this.expireAfterWriteMillis, that.expireAfterWriteMillis);
                    }
                    - return false;
            }

            @Override
            public String toString() {
                    - return Objects.toStringHelper(this)
                            - .add("expireAfterAccessMillis", expireAfterAccessMillis)
                            - .add("expireAfterWriteMillis", expireAfterWriteMillis)
                            - .toString();
            }
    }
```
```java
+static class DurationSpec {
    + private final long duration;
    + private final TimeUnit unit;

    + private DurationSpec(long duration, TimeUnit unit) {
        + this.duration = duration;
        + this.unit = unit;
    }

    + public static DurationSpec of(long duration, TimeUnit unit) {
        + return new DurationSpec(duration, unit);
    }

    @Override
    public int hashCode() {
        + return Objects.hashCode(duration, unit);
    }

    @Override
    public boolean equals(Object o) {
        + if (o instanceof DurationSpec) {
            DurationSpec that = (DurationSpec) o;
            + return unit.toNanos(duration) == that.unit.toNanos(that.duration);
        }
        + return false;
    }

    @Override
    public String toString() {
        + return Objects.toStringHelper(this)
            + .add("duration", duration)
            + .add("unit", unit)
            .toString();
    }
```
```java
static class ExpirationSpec {
    @Nullable
    private final Long expireAfterAccessMillis;
    @Nullable
    private final Long expireAfterWriteMillis;

    private ExpirationSpec(Long expireAfterAccessMillis, Long expireAfterWriteMillis) {
        Preconditions.checkNotNull(
            expireAfterAccessMillis == null || expireAfterWriteMillis == null);
        this.expireAfterAccessMillis = expireAfterAccessMillis;
        this.expireAfterWriteMillis = expireAfterWriteMillis;
    }

    public static ExpirationSpec afterAccess(long afterAccess, TimeUnit unit) {
        return new ExpirationSpec(unit.toMillis(afterAccess), null);
    }

    public static ExpirationSpec afterWrite(long afterWrite, TimeUnit unit) {
        return new ExpirationSpec(null, unit.toMillis(afterWrite));
    }

    @Override
    public int hashCode() {
        return Objects.hashCode(expireAfterAccessMillis, expireAfterWriteMillis);
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof ExpirationSpec) {
            ExpirationSpec that = (ExpirationSpec) o;
            return Objects.equals(this.expireAfterAccessMillis, that.expireAfterAccessMillis) &&
                Objects.equals(this.expireAfterWriteMillis, that.expireAfterWriteMillis);
        }
        return false;
    }

    @Override
    public String toString() {
        return Objects.toStringHelper(this)
            .add("expireAfterAccessMillis", expireAfterAccessMillis)
            .add("expireAfterWriteMillis", expireAfterWriteMillis)
            .toString();
    }
}
```
```java
static class DurationSpec {
    private final long duration;
    private final TimeUnit unit;

    private DurationSpec(long duration, TimeUnit unit) {
        this.duration = duration;
        this.unit = unit;
    }

    public static DurationSpec of(long duration, TimeUnit unit) {
        return new DurationSpec(duration, unit);
    }

    @Override
    public int hashCode() {
        return Objects.hashCode(duration, unit);
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof DurationSpec) {
            DurationSpec that = (DurationSpec) o;
            return unit.toNanos(duration) == that.unit.toNanos(that.duration);
        }
        return false;
    }

    @Override
    public String toString() {
        return Objects.toStringHelper(this)
                .add("duration", duration)
                .add("unit", unit)
                .toString();
    }
}
```
```java
static class ExpirationSpec {
    @Nullable
    private final Long expireAfterAccessMillis;
    @Nullable
    private final Long expireAfterWriteMillis;

    private ExpirationSpec(Long expireAfterAccessMillis, Long expireAfterWriteMillis) {
        Preconditions.checkNotNull(expireAfterAccessMillis == null || expireAfterWriteMillis == null);
        this.expireAfterAccessMillis = expireAfterAccessMillis;
        this.expireAfterWriteMillis = expireAfterWriteMillis;
    }

    public static ExpirationSpec afterAccess(long afterAccess, TimeUnit unit) {
        return new ExpirationSpec(unit.toMillis(afterAccess), null);
    }

    public static ExpirationSpec afterWrite(long afterWrite, TimeUnit unit) {
        return new ExpirationSpec(null, unit.toMillis(afterWrite));
    }

    @Override
    public int hashCode() {
        return Objects.hash(expireAfterAccessMillis, expireAfterWriteMillis);
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof ExpirationSpec) {
            ExpirationSpec that = (ExpirationSpec) o;
            return Objects.equals(this.expireAfterAccessMillis, that.expireAfterAccessMillis)
                    && Objects.equals(this.expireAfterWriteMillis, that.expireAfterWriteMillis);
        }

        return false;
    }

    @Override
    public String toString() {
        return Objects.toStringHelper(this)
                .add("expireAfterAccessMillis", expireAfterAccessMillis)
                .add("expireAfterWriteMillis", expireAfterWriteMillis)
                .toString();
    }
}
```
```java
static class DurationSpec {

    private final long duration;
    private final TimeUnit unit;

    private DurationSpec(long duration, TimeUnit unit) {
        this.duration = duration;
        this.unit = unit;
    }

    public static DurationSpec of(long duration, TimeUnit unit) {
        return new DurationSpec(duration, unit);
    }

    @Override
    public int hashCode() {
        return Objects.hashCode(duration, unit);
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof DurationSpec) {
            DurationSpec that = (DurationSpec) o;
            return unit.toNanos(duration) == that.unit.toNanos(that.duration);
        }
        return false;
    }

    @Override
    public String toString() {
        return Objects.toStringHelper(this)
            .add("duration", duration)
            .add("unit", unit)
            .toString();
    }
}
```
static class ExpirationSpec {
    @Nullable
    private final Long expireAccessMillis;
    @Nullable
    private final Long expireWriteMillis;

    private ExpirationSpec(Long expireAfterAccessMillis, Long expireAfterWriteMillis) {
        Preconditions.checkNotNull(
            expireAfterAccessMillis == null || expireAfterWriteMillis == null);
        this.expireAccessMillis = expireAfterAccessMillis;
        this.expireWriteMillis = expireAfterWriteMillis;
    }

    public static ExpirationSpec afterAccess(long afterAccess, TimeUnit unit) {
        return new ExpirationSpec(unit.toMillis(afterAccess), null);
    }

    public static ExpirationSpec afterWrite(long afterWrite, TimeUnit unit) {
        return new ExpirationSpec(null, unit.toMillis(afterWrite));
    }

    @Override
    public int hashCode() {
        return Objects.hash(expireAccessMillis, expireWriteMillis);
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof ExpirationSpec) {
            ExpirationSpec that = (ExpirationSpec) o;
            return Objects.equals(this.expireAccessMillis, that.expireAccessMillis) &&
                   Objects.equals(this.expireWriteMillis, that.expireWriteMillis);
        }
        return false;
    }

    @Override
    public String toString() {
        return Objects.toStringHelper(this)
            .add("expireAccessMillis", expireAccessMillis)
            .add("expireWriteMillis", expireWriteMillis)
            .toString();
    }
}
Sample 10 GumTree Modified

```java
static class DurationSpec {
    private final long duration;
    private final TimeUnit unit;

    private DurationSpec(long duration, TimeUnit unit) {
        this.duration = duration;
        this.unit = unit;
    }

    public static DurationSpec of(long duration, TimeUnit unit) {
        return new DurationSpec(duration, unit);
    }

    @Override
    public int hashCode() {
        return Objects.hash(duration, unit);
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof DurationSpec) {
            DurationSpec that = (DurationSpec) o;
            return unit.toNanos(duration) == that.unit.toNanos(that.duration);
        }
        return false;
    }

    @Override
    public String toString() {
        return Objects.toStringHelper(this)
            .add("duration", duration)
            .add("unit", unit)
            .toString();
    }
}
```
public void expandFolds(int foldLevel, boolean update) {
    if(buffer.getFoldHandler() instanceof IndentFoldHandler)
        foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;

    showLineRange(0,buffer.getLineCount() - 1);

    int leastFolded = -1;
    int firstInvisible = 0;

    for(int i = 0; i < buffer.getLineCount(); i++)
    {
        if (leastFolded == -1 || buffer.getFoldLevel(i) < leastFolded)
        {
            leastFolded = buffer.getFoldLevel(i);
        }

        if (buffer.getFoldLevel(i) < foldLevel ||
            buffer.getFoldLevel(i) == leastFolded)
        {
            if(firstInvisible != i)
            {
                hideLineRange(firstInvisible,
                                i - 1);
            }

            firstInvisible = i + 1;
        }
    }

    if(firstInvisible != buffer.getLineCount())
        hideLineRange(firstInvisible,buffer.getLineCount() - 1);

    notifyScreenLineChanges();
    if(update && textarea.getDisplayManager() == this)
    {
        textarea.foldStructureChanged();
    }
}
public void expandFolds(int foldLevel, boolean update) {
    if (buffer.getFoldHandler() instanceof IndentFoldHandler)
        foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;

    int lineCount = buffer.getLineCount();
    int end = lineCount - 1;
    showLineRange(0, end);

    int leastFolded = -1;
    int firstInvisible = 0;

    for (int i = 0; i < lineCount; i++) {
        int level = buffer.getFoldLevel(i);
        if (leastFolded == -1 || level < leastFolded)
            leastFolded = level;
    }

    if (level < foldLevel || level == leastFolded)
        if (firstInvisible != i)
            hideLineRange(firstInvisible, i - 1);
    firstInvisible = i + 1;
}

if (firstInvisible != lineCount)
    hideLineRange(firstInvisible, end);

notifyScreenLineChanges();
if (update && textarea.getDisplayManager() == this)
    { textarea.foldStructureChanged(); }
}
Sample 11 GitHub Original

```java
... @@ -1,22 +1,24 @@ @@
1 public void expandFolds(int foldLevel, boolean update)
2 {
3     if(buffer.getFoldHandler() instanceof IndentFoldHandler)
4         foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;
5
6     - showLineRange(0,buffer.getLineCount() - 1);
7
8     int leastFolded = -1;
9     int firstInvisible = 0;
10
11    - for(int i = 0; i < buffer.getLineCount(); i++)
12       {
13           - if (leastFolded == -1 || buffer.getFoldLevel(i) < leastFolded)
14             {
15               leastFolded = buffer.getFoldLevel(i);
16             }
17
18           - if (buffer.getFoldLevel(i) < foldLevel ||
19               buffer.getFoldLevel(i) == leastFolded)
20             {
21               if(firstInvisible != i)
22                 {
23                     hideLineRange(firstInvisible,
24                         i - 1);
25                 }
26               firstInvisible = i + 1;
27             }
28         }
29
30    - if(firstInvisible != buffer.getLineCount())
31       - hideLineRange(firstInvisible,buffer.getLineCount() - 1);
32
33     notifyScreenLineChanges();
34     if(update && textarea.getDisplayManager() == this)
35       {
36         textarea.foldStructureChanged();
37     }
38 }
```
```java
public void expandFolds(int foldLevel, boolean update) {
    if (buffer.getFoldHandler() instanceof IndentFoldHandler) {
        foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;
    }

    int lineCount = buffer.getLineCount();
    int end = lineCount - 1;
    showLineRange(0, end);

    int leastFolded = -1;
    int first Invisible = 0;

    for (int i = 0; i < lineCount; i++) {
        int level = buffer.getFoldLevel(i);
        if (leastFolded == -1 || level < leastFolded) {
            leastFolded = level;
        }

        if (level < foldLevel || level == leastFolded) {
            if (firstInvisible != i)
                hideLineRange(firstInvisible, i - 1);
            firstInvisible = i + 1;
        }
    }

    if (firstInvisible != lineCount)
        hideLineRange(firstInvisible, end);

    notifyScreenLineChanges();
    if (update & displayManager() == this) {
        textArea.foldStructureChanged();
    }
}
```
public void expandFolds(int foldLevel, boolean update) {
    if(buffer.getFoldHandler() instanceof IndentFoldHandler) {
        foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;
    }

    showLineRange(0, buffer.getLineCount() - 1);

    int leastFolded = -1;
    int firstInvisible = 0;

    for(int i = 0; i < buffer.getLineCount(); i++) {
        if (leastFolded == -1 || buffer.getFoldLevel(i) < leastFolded) {
            leastFolded = buffer.getFoldLevel(i);
        }

        if (buffer.getFoldLevel(i) < foldLevel || buffer.getFoldLevel(i) == leastFolded) {
            if (firstInvisible != i) {
                hideLineRange(firstInvisible, i - 1);
            }

            firstInvisible = i + 1;
        }
    }

    if(firstInvisible != buffer.getLineCount()) {
        hideLineRange(firstInvisible, buffer.getLineCount() - 1);
    }

    notifyScreenLineChanges();
    if(update && textarea.getDisplayManager() == this) {
        textarea.foldStructureChanged();
    }
}
Sample 11 Mergely Modified

```java
public void expandFolds(int foldLevel, boolean update)
{
    if(buffer.getFoldHandler() instanceof IndentFoldHandler)
        foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;

    int lineCount = buffer.getLineCount();
    int end = lineCount - 1;
    showLineRange(0,end);

    int leastFolded = -1;
    int firstInvisible = 0;

    for(int i = 0; i < lineCount; i++)
    {
        int level = buffer.getFoldLevel(i);
        if (leastFolded == -1 || level < leastFolded)
        {
            leastFolded = level;
        }
    }

    if (level < foldLevel || level == leastFolded)
    {
        if(firstInvisible != i)
        {
            hideLineRange(firstInvisible, i - 1);
        }
        firstInvisible = i + 1;
    }

    if(firstInvisible != lineCount)
        hideLineRange(firstInvisible,end);

    notifyScreenLineChanges();
    if(update && textView.getDisplayManager() == this)
    {
        textView.foldStructureChanged();
    }
}
```
public void expandFolds(int foldLevel, boolean update) {
    if(buffer.getFoldHandler() instanceof IndentFoldHandler)
        foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;

    showLineRange(0, buffer.getLineCount() - 1);
    int leastFolded = -1;
    int firstInvisible = 0;

    for(int i = 0; i < buffer.getLineCount(); i++)
    {
        if (leastFolded == -1 || buffer.getFoldLevel(i) < leastFolded)
        {
            leastFolded = buffer.getFoldLevel(i);
        }

        if (buffer.getFoldLevel(i) < foldLevel ||
                        buffer.getFoldLevel(i) == leastFolded)
        {
            if(firstInvisible != i)
            {
                hideLineRange(firstInvisible, i - 1);
            }
            firstInvisible = i + 1;
        }
    }

    if(firstInvisible != buffer.getLineCount())
        hideLineRange(firstInvisible, buffer.getLineCount() - 1);

    notifyScreenLineChanges();
    if(update && textarea.getDisplayManager() == this)
    {
        textarea.foldStructureChanged();
    }
}
public void expandFolds(int foldLevel, boolean update) {
    if (buffer.getFoldHandler() instanceof IndentFoldHandler)
        foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;

    int lineCount = buffer.getLineCount();
    int end = lineCount - 1;
    showLineRange(0, end);

    int leastFolded = -1;
    int firstInvisible = 0;

    for (int i = 0; i < lineCount; i++) {
        int level = buffer.getFoldLevel(i);
        if (leastFolded == -1 || level < leastFolded)
            leastFolded = level;
    }

    if (level < foldLevel ||
        level == leastFolded)
        if (firstInvisible != i)
            hideLineRange(firstInvisible, i - 1);
        firstInvisible = i + 1;

    if (firstInvisible != lineCount)
        hideLineRange(firstInvisible, end);

    notifyScreenLineChanges();
    if (update && textarea.getDisplayManager() == this)
        textarea.foldStructureChanged();
}
public void expandFolds(int foldLevel, boolean update)
{
    if(buffer.getFoldHandler() instanceof IndentFoldHandler)
        foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;

    showLineRange(0, buffer.getLineCount() - 1);

    int leastFolded = -1;
    int firstInvisible = 0;

    for(int i = 0; i < buffer.getLineCount(); i++)
    {
        if (leastFolded == -1 || buffer.getFoldLevel(i) < leastFolded)
        {
            leastFolded = buffer.getFoldLevel(i);
        }

        if (buffer.getFoldLevel(i) < foldLevel ||
            buffer.getFoldLevel(i) == leastFolded)
        {
            if(firstInvisible != i)
            {
                hideLineRange(firstInvisible, i - 1);
            }
            firstInvisible = i + 1;
        }
    }

    if(firstInvisible != buffer.getLineCount())
        hideLineRange(firstInvisible, buffer.getLineCount() - 1);

    notifyScreenLineChanges();
    if(update && textarea.getDisplayManager() == this)
    {
        textarea.foldStructureChanged();
    }
}
public void expandFolds(int foldLevel, boolean update)
{
    if(buffer.getFoldHandler() instanceof IndentFoldHandler)
        foldLevel = (foldLevel - 1) * buffer.getIndentSize() + 1;

    int lineCount = buffer.getLineCount();
    int end = lineCount - 1;
    showLineRange(0,end);

    int leastFolded = -1;
    int firstInvisible = 0;

    for(int i = 0; i < lineCount; i++)
    {
        int level = buffer.getFoldLevel(i);
        if (leastFolded == -1 || level < leastFolded)
        {
            leastFolded = level;
        }
    }

    if (level < foldLevel || level == leastFolded)
    {
        if(firstInvisible != i)
        {
            hideLineRange(firstInvisible,
                          i - 1);
        }
        firstInvisible = i + 1;
    }

    if(firstInvisible != lineCount)
        hideLineRange(firstInvisible,end);

    notifyScreenLineChanges();
    if(update && textField.getDisplayManager() == this)
    {
        textField.foldStructureChanged();
    }
}
```c
void physDown(int amount, int screenAmount)
{
    skew = 0;

    if(!isLineVisible(physicalLine))
    {
        int lastVisibleLine = getLastVisibleLine();
        if(physicalLine > lastVisibleLine)
            physicalLine = lastVisibleLine;
        else
        {
            int nextPhysicalLine = getNextVisibleLine(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            scrollLine += getScreenLineCount(physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    for(;;)
    {
        int nextPhysicalLine = getNextVisibleLine(
            physicalLine);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physicalLine + amount)
            break;
        else
        {
            scrollLine += getScreenLineCount(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    if(screenAmount != 0)
    {
        if(screenAmount < 0)
            scrollUp(-screenAmount);
        else
            scrollDown(screenAmount);
    }
}
```
void physDown(int amount, int screenAmount)
{
    skew = 0;

    if(!isLineVisible(physicalLine))
    {
        int lastVisibleLine = getLastVisibleLine();
        if(physicalLine > lastVisibleLine)
            physicalLine = lastVisibleLine;
        else
        {
            int nextPhysicalLine = getNextVisibleLine(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            scrollLine += getScreenLineCount(physicalLine); physicalLine = nextPhysicalLine;
        }
    }

    for(;;)
    {
        int nextPhysicalLine = getNextVisibleLine(physicalLine);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physicalLine + amount)
            break;
        else
        {
            scrollLine += getScreenLineCount(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    if(screenAmount < 0)
        scrollUp(-screenAmount);
    else if(screenAmount > 0)
        scrollDown(screenAmount);
}
```c
void physDown(int amount, int screenAmount)
{
    skew = 0;
    
    if(!isLineVisible(physcalline))
    {
        int lastVisibleLine = getLastVisibleLine();
        if(physcalline > lastVisibleLine)
            physcalline = lastVisibleLine;
        else
        {
            int nextPhysicalLine = getNextVisibleLine(physcalline);
            amount -= (nextPhysicalLine - physcalline);
            scrollLine += getScreenLineCount(physcalline);
            physcalline = nextPhysicalLine;
        }
    }
    
    for(;;)
    {
        int nextPhysicalLine = getNextVisibleLine(physcalline);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physcalline + amount)
            break;
        else
        {
            scrollLine += getScreenLineCount(physcalline);
            amount -= (nextPhysicalLine - physcalline);
            physcalline = nextPhysicalLine;
        }
    }
}
```

```java
void physDown(int amount, int screenAmount)
{
    skew = 0;

    if(!isLineVisible(physicalLine))
    {
        int lastVisibleLine = getLastVisibleLine();
        if(physicalline > lastVisibleLine)
            physicalline = lastVisibleLine;
        else
            {
                int nextPhysicalLine = getNextVisibleLine(physicalline);
                amount -= (nextPhysicalLine - physicalline);
                scrollLine += getScreenLineCount(physicalline);
                physicalline = nextPhysicalLine;
            }
    }

    for(;;)
    {
        int nextPhysicalLine = getNextVisibleLine(physicalline);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physicalline + amount)
            break;
        else
            {
                scrollLine += getScreenLineCount(physicalline);
                amount -= (nextPhysicalLine - physicalline);
                physicalline = nextPhysicalLine;
            }
    }

    if(screenAmount < 0)
        scrollUp(-screenAmount);
    else if(screenAmount > 0)
        scrollDown(screenAmount);
}
```
void physDown(int amount, int screenAmount)
{
    skew = 0;

    if(!isLineVisible(physicalLine))
    {
        int lastVisibleLine = getLastVisibleLine();
        if(physicalLine > lastVisibleLine)
            physicalLine = lastVisibleLine;
        else
            {
                int nextPhysicalLine = getNextVisibleLine(physicalLine);
                amount -= (nextPhysicalLine - physicalLine);
                scrollLine += getScreenLineCount(physicalLine);
                physicalLine = nextPhysicalLine;
            }
    }

    for(;;)
    {
        int nextPhysicalLine = getNextVisibleLine(
            physicalLine);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physicalLine + amount)
            break;
        else
            {
                scrollLine += getScreenLineCount(physicalLine);
                amount -= (nextPhysicalLine - physicalLine);
                physicalLine = nextPhysicalLine;
            }
    }

    if(screenAmount != -0)
    {
        if(screenAmount < 0)
            scrollUp(-screenAmount);
        else
            scrollDown(screenAmount);
    }
void physDown(int amount, int screenAmount) {
    skew = 0;

    if(!isLineVisible(physicalLine)) {
        int lastVisibleLine = getLastVisibleLine();
        if(physicalLine > lastVisibleLine)
            physicalLine = lastVisibleLine;
        else {
            int nextPhysicalLine = getNextVisibleLine(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            scrollLine += getScreenLineCount(physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    for(;;) {
        int nextPhysicalLine = getNextVisibleLine(physicalLine);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physicalLine + amount)
            break;
        else {
            scrollLine += getScreenLineCount(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    if(screenAmount < 0)
        scrollUp(-screenAmount);
    else if(screenAmount > 0)
        scrollDown(screenAmount);
void physDown(int amount, int screenAmount)
{
    skew = 0;

    if(!isLineVisible(physicalLine))
    {
        int lastVisibleLine = getLastVisibleLine();
        if(physicalLine > lastVisibleLine)
            physicalLine = lastVisibleLine;
        else
        {
            int nextPhysicalLine = getNextVisibleLine(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            scrollLine += getScreenLineCount(physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    for(;;)
    {
        int nextPhysicalLine = getNextVisibleLine(physicalLine);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physicalLine + amount)
            break;
        else
        {
            scrollLine += getScreenLineCount(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    if(screenAmount != 0)
    {
        if(screenAmount < 0)
            scrollUp(-screenAmount);
        else
            scrollDown(screenAmount);
    }
Sample 12 srcDiff Modified

```c
void physDown(int amount, int screenAmount)
{
    skew = 0;

    if(!isLineVisible(physicalLine))
    {
        int lastVisibleLine = getLastVisibleLine();
        if(physicalLine > lastVisibleLine)
            physicalLine = lastVisibleLine;
        else
        {
            int nextPhysicalLine = getNextVisibleLine(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            scrollLine += getScreenLineCount(physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    for(;;)
    {
        int nextPhysicalLine = getNextVisibleLine(
                              physicalLine);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physicalLine + amount)
            break;
        else
        {
            scrollLine += getScreenLineCount(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    if(screenAmount < 0)
        scrollUp(-screenAmount);
    else if(screenAmount > 0)
        scrollDown(screenAmount);
}
```
Sample 12 GumTree Original

```c
void physDown(int amount, int screenAmount)
{
    skew = 0;

    if(!isLineVisible(physicalLine))
    {
        int lastVisibleLine = getLastVisibleLine();
        if(physicalLine > lastVisibleLine)
            physicalLine = lastVisibleLine;
        else
        {
            int nextPhysicalLine = getNextVisibleLine(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            scrollLine += getScreenLineCount(physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    for(;;)
    {
        int nextPhysicalLine = getNextVisibleLine(
            physicalLine);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physicalLine + amount)
            break;
        else
        {
            scrollLine += getScreenLineCount(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    if(screenAmount != 0)
    {
        if(screenAmount < 0)
            scrollUp(-screenAmount);
        else
            scrollDown(screenAmount);
    }
}
```
Sample 12 GumTree Modified

void physDown(int amount, int screenAmount)
{
    skew = 0;

    if(!isLineVisible(physicalLine))
    {
        int lastVisibleLine = getLastVisibleLine();
        if(physicalLine > lastVisibleLine)
            physicalLine = lastVisibleLine;
        else
        {
            int nextPhysicalLine = getNextVisibleLine(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            scrollLine += getScreenLineCount(physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    for(;;)
    {
        int nextPhysicalLine = getNextVisibleLine(physicalLine);
        if(nextPhysicalLine == -1)
            break;
        else if(nextPhysicalLine > physicalLine + amount)
            break;
        else
        {
            scrollLine += getScreenLineCount(physicalLine);
            amount -= (nextPhysicalLine - physicalLine);
            physicalLine = nextPhysicalLine;
        }
    }

    if(screenAmount < 0)
        scrollUp(-screenAmount);
    else if(screenAmount > 0)
        scrollDown(screenAmount);
}
```java
public static Long tryParse(String string) {
    if (checkNotNull(string).isEmpty()) {
        return null;
    }
    boolean negative = string.charAt(0) == '-';
    int index = negative ? 1 : 0;
    if (index == string.length()) {
        return null;
    }
    int digit = string.charAt(index++) - '0';
    if (digit < 0 || digit > 9) {
        return null;
    }
    long accum = -digit;
    while (index < string.length()) {
        digit = string.charAt(index++) - '0';
        if (digit < 0 || digit > 9 || accum < Long.MIN_VALUE / 10) {
            return null;
        }
        accum *= 10;
        if (accum < Long.MIN_VALUE + digit) {
            return null;
        }
        accum -= digit;
    }
    if (negative) {
        return accum;
    } else if (accum == Long.MIN_VALUE) {
        return null;
    } else {
        return -accum;
    }
}
```
public static Long tryParse(String string, int radix) {
    if (checkNotNull(string).isEmpty()) {
        return null;
    }
    if (radix < Character.MIN_RADIX || radix > Character.MAX_RADIX) {
        throw new IllegalArgumentException("radix must be between MIN_RADIX and MAX_RADIX but was " + radix);
    }
    boolean negative = string.charAt(0) == '-';
    int index = negative ? 1 : 0;
    if (index == string.length()) {
        return null;
    }
    int digit = digit(string.charAt(index++));
    if (digit < 0 || digit >= radix) {
        return null;
    }
    long accum = -digit;

    long cap = Long.MIN_VALUE / radix;

    while (index < string.length()) {
        digit = digit(string.charAt(index++));
        if (digit < 0 || digit >= radix || accum < cap) {
            return null;
        }
        accum *= radix;
        if (accum < Long.MIN_VALUE + digit) {
            return null;
        }
        accum -= digit;
    }

    if (negative) {
        return accum;
    } else if (accum == Long.MIN_VALUE) {
        return null;
    } else {
        return -accum;
    }
}
```java
-@-1,26 +1,33 @@

@Beta
@Nullable
@CheckForNull

-@-public static Long tryParse(String string) {
      if (checkWotNull(string).isEmpty()) {
          return null;
      }

-@boolean negative = string.charAt(0) == '-';
-@int index = negative ? 1 : 0;
-@if (index == string.length()) {
          return null;
      }

-@-int digit = string.charAt(index++) == '0';
-@-if (digit < 0 || digit > '9') {
          return null;
      }

-@long accum = -digit;

-@while (index < string.length()) {
-@-int digit = string.charAt(index++) == '0';
-@-if (digit < 0 || digit > '9' || accum < Long.MIN_VALUE / 10) {
                      return null;
-@      }

-@-accum *= 10;
-@if (accum < Long.MIN_VALUE + digit) {
          return null;
      }

-@accum -= digit;

-@if (negative) {
-@    return accum;
-@} else if (accum == Long.MIN_VALUE) {
          return null;
-@} else {
          return -accum;
-@}
```
290

Sample 13 GitHub Modified

```java
@Beta
@Nullables
@CheckForNull
public static long tryParse(String string, int radix) {
    if (checkNotNull(string).isEmpty()) {
        return null;
    }

    if (radix < Character.MIN_RADIX || radix > Character.MAX_RADIX) {
        throw new IllegalArgumentException("radix must be between MIN_RADIX and MAX_RADIX but was "+ radix);
    }

    boolean negative = string.charAt(0) == '-';
    int index = negative ? 1 : 0;
    if (index == string.length()) {
        return null;
    }

    int digit = digit(string.charAt(index++));
    if (digit < 0 || digit >= radix) {
        return null;
    }

    long accum = -digit;

    while (index < string.length()) {
        digit = digit(string.charAt(index++));
        if (digit < 0 || digit >= radix || accum < cap) {
            return null;
        }
        accum *= radix;
    }

    if (accum < Long.MIN_VALUE + digit) {
        return null;
    }

    accum -= digit;

    if (negative) {
        return accum;
    } else if (accum == Long.MIN_VALUE) {
        return null;
    } else {
        return -accum;
    }
}
```
public static Long tryParse(String string) {
    if (checkNotNull(string).isEmpty()) {
        return null;
    }
    boolean negative = string.charAt(0) == '-'
    int index = negative ? 1 : 0;
    if (index == string.length()) {
        return null;
    }
    int digit = string.charAt(index++) - '0';
    if (digit < 0 || digit > 9) {
        return null;
    }
    long accum = -digit;
    while (index < string.length()) {
        digit = string.charAt(index++) - '0';
        if (digit < 0 || digit > 9 || accum < Long.MIN_VALUE / 10) {
            return null;
        }
        accum *= 10;
        if (accum < Long.MIN_VALUE + digit) {
            return null;
        }
        accum -= digit;
    }
    if (negative) {
        return accum;
    } else if (accum == Long.MIN_VALUE) {
        return null;
    } else {
        return -accum;
    }
}
@Beta
@Nullable
@CheckForNull
public static Long tryParse(String string, int radix) {
    if (checkNotNull(string).isEmpty()) {
        return null;
    }
    if (radix < Character.MIN_RADIX || radix > Character.MAX_RADIX) {
        throw new IllegalArgumentException(
            "radix must be between MIN_RADIX and MAX_RADIX but was " + radix);
    }
    boolean negative = string.charAt(0) == '-';
    int index = negative ? 1 : 0;
    if (index == string.length()) {
        return null;
    }
    int digit = digit(string.charAt(index++));
    if (digit < 0 || digit >= radix) {
        return null;
    }
    long accum = -digit;
    long cap = Long.MIN_VALUE / radix;
    while (index < string.length()) {
        digit = digit(string.charAt(index++));
        if (digit < 0 || digit >= radix || accum < cap) {
            return null;
        }
        accum *= radix;
        if (accum < Long.MIN_VALUE + digit) {
            return null;
        }
        accum -= digit;
    }
    if (negative) {
        return accum;
    } else if (accum == Long.MIN_VALUE) {
        return null;
    } else {
        return -accum;
    }
}
```java
@Beta
@Nullable
@CheckForNull
public static Long tryParse(String string) {
    if (checkNotNull(string).isEmpty()) {
        return null;
    }

    boolean negative = string.charAt(0) == '-';
    int index = negative ? 1 : 0;
    if (index == string.length()) {
        return null;
    }
    int digit = string.charAt(index++) == '0';
    if (digit < 0 || digit > 9) {
        return null;
    }
    long accum = -digit;

    while (index < string.length()) {
        digit = string.charAt(index++) == '0';
        if (digit < 0 || digit > 9 || accum < Long.MIN_VALUE / 10) {
            return null;
        }
        accum *= 10;
        if (accum < Long.MIN_VALUE + digit) {
            return null;
        }
        accum -= digit;
    }
    if (negative) {
        return accum;
    } else if (accum == Long.MIN_VALUE) {
        return null;
    } else {
        return -accum;
    }
}
Modified

```java
@Beta
@Nullable
@CheckForNull
public static Long tryParse(String string, int radix) {
    if (checkNotNull(string).isEmpty()) {
        return null;
    }

    if (radix < Character.MIN_RADIX || radix > Character.MAX_RADIX) {
        throw new IllegalArgumentException("radix must be between MIN_RADIX and MAX_RADIX but was " + radix);
    }

    boolean negative = string.charAt(0) == '-';
    int index = negative ? 1 : 0;
    if (index == string.length()) {
        return null;
    }

    int digit = digit(string.charAt(index++));
    if (digit < 0 || digit >= radix) {
        return null;
    }

    long accum = -digit;

    long cap = Long.MIN_VALUE / radix;

    while (index < string.length()) {
        digit = digit(string.charAt(index++));
        if (digit < 0 || digit >= radix || accum < cap) {
            return null;
        }
        accum *= radix;
        if (accum < Long.MIN_VALUE + digit) {
            return null;
        }
        accum -= digit;
    }

    if (negative) {
        return accum;
    } else if (accum == Long.MIN_VALUE) {
        return null;
    } else {
        return -accum;
    }
}
```
Sample 13 GumTree Original

```java
@Beta
@Nullable
@CheckForNull
public static Long tryParse(String string) {
    if (checkNotNull(string).isEmpty()) {
        return null;
    }
    boolean negative = string.charAt(0) == '-';
    int index = negative ? 1 : 0;
    if (index == string.length()) {
        return null;
    }
    int digit = string.charAt(index++) - '0';
    if (digit < 0 || digit > 9) {
        return null;
    }
    long accum = -digit;
    while (index < string.length()) {
        digit = string.charAt(index++) - '0';
        if (digit < 0 || digit > 9 || accum < Long.MIN_VALUE / 10) {
            return null;
        }
        accum *= 10;
        if (accum < Long.MIN_VALUE + digit) {
            return null;
        }
        accum -= digit;
    }
    if (negative) {
        return accum;
    } else if (accum == Long.MIN_VALUE) {
        return null;
    } else {
        return -accum;
    }
}
```
Sample 13 GumTree Modified

```java
@Beta
@Nullable
@CheckForNull
public static Long tryParse(String string, int radix) {
    if (checkNotNull(string).isEmpty()) {
        return null;
    }
    if (radix < Character.MIN_RADIX || radix > Character.MAX_RADIX) {
        throw new IllegalArgumentException("radix must be between MIN_RADIX and MAX_RADIX but was " + radix);
    }
    boolean negative = string.charAt(0) == '-';
    int index = negative ? 1 : 0;
    if (index == string.length()) {
        return null;
    }
    int digit = digit(string.charAt(index++));
    if (digit < 0 || digit >= radix) {
        return null;
    }
    long accum = -digit;

    long cap = Long.MIN_VALUE / radix;

    while (index < string.length()) {
        digit = digit(string.charAt(index++));
        if (digit < 0 || digit >= radix || accum < cap) {
            return null;
        }
        accum *= radix;
        if (accum < Long.MIN_VALUE + digit) {
            return null;
        }
        accum -= digit;
    }

    if (negative) {
        return accum;
    } else if (accum == Long.MIN_VALUE) {
        return null;
    } else {
        return -accum;
    }
}
```
Sample 14 Source Original

<table>
<thead>
<tr>
<th>Original</th>
</tr>
</thead>
</table>
| public ImmutableSet<ClassInfo> getClasses(Package pkg) {
  return getClasses(pkg.getName());
} |

Sample 14 Source Modified

<table>
<thead>
<tr>
<th>Modified</th>
</tr>
</thead>
</table>
| public ImmutableSet<ClassInfo> getClassesRecursive(String packageName) {
  checkNotNull(packageName);
  String packagePrefix = packageName + '.';
  ImmutableSet.Builder<ClassInfo> builder = ImmutableSet.builder();
  for (ClassInfo classInfo : classes) {
    if (classInfo.getName().startsWith(packagePrefix)) {
      builder.add(classInfo);
    }
  }
  return builder.build();
} |
Sample 14 GitHub Original

```java
@-1,+1
-@ -public ImmutableSet<ClassInfo> getClasses(Package pkg) {
  - return getClasses(pkg.getName());
}
```

Sample 14 GitHub Modified

```java
+public ImmutableSet<ClassInfo> getClassesRecursive(String packageName) {
  + checkNotNull(packageName);
  + String packagePrefix = packageName + '.';
  + ImmutableSet.Builder<ClassInfo> builder = ImmutableSet.builder();
  + for (ClassInfo classInfo : classes) {
    + if (classInfo.getName().startsWith(packagePrefix)) {
      + builder.add(classInfo);
    + }
  + } return builder.build();
}
Sample 14 Mergely Original

```java
public ImmutableSet<ClassInfo> getClasses(Package pkg) {
    return getClasses(pkg.getName());
}
```

Sample 14 Merely Modified

```java
public ImmutableSet<ClassInfo> getClassesRecursive(String packageName) {
    checkNotNull(packageName);
    String packagePrefix = packageName + '.';
    ImmutableSet.Builder<ClassInfo> builder = ImmutableSet.builder();
    for (ClassInfo classInfo : classes) {
        if (classInfo.getName().startsWith(packagePrefix)) {
            builder.add(classInfo);
        }
    }
    return builder.build();
}
```
Sample 14 srcDiff Original

```
Original

public ImmutableSet<ClassInfo> getClasses(Package pkg) {
    return getClasses(pkg.getName());
}
```

Sample 14 srcDiff Modified

```
Modified

public ImmutableSet<ClassInfo> getClassesRecursive(String packageName) {
    checkNotNull(packageName);
    String packagePrefix = packageName + '.';
    ImmutableSet.Builder<ClassInfo> builder = ImmutableSet.builder();
    for (ClassInfo classInfo : classes) {
        if (classInfo.getName().startsWith(packagePrefix)) {
            builder.add(classInfo);
        }
    }
    return builder.build();
}
```
Sample 14 GumTree Original

```java
public ImmutableSet<ClassInfo> getClasses(Package pkg) {
    return getClasses(pkg.getName());
}
```

Sample 14 GumTree Modified

```java
public ImmutableSet<ClassInfo> getClassesRecursive(String packageName) {
    checkNotNull(packageName);
    String packagePrefix = packageName + '.';
    ImmutableSet.Builder<ClassInfo> builder = ImmutableSet.builder();
    for (ClassInfo classInfo : classes) {
        if (classInfo.getName().startsWith(packagePrefix)) {
            builder.add(classInfo);
        }
    }
    return builder.build();
}
```
REFERENCES


Information", in Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 493-504.


