Towards Convenient Management of Software Clone Codes in Practice: An Integrated Approach

Md Sharif Uddin Chanchal K. Roy Kevin A. Schneider
Dept. of Computer Science
University of Saskatchewan
Saskatoon, Canada
{s.uddin, chanchal.roy, kevin.schneider}@usask.ca

ABSTRACT

Software code cloning is inevitable during software development and unmanaged cloning practice can create substantial problems for software maintenance and evolution. Current research in the area software clones includes, but is not limited to: finding ways to manage clones; gaining more control over clone generation; and, studying clone evolution and its effects on the evolution of software. In this study, we investigate tools and techniques for detecting, managing, and understanding the evolution of clones, as well as design a convenient tool to make those techniques available to a developer's software development environment. Towards the goal of promoting the practical use of code clone research and to provide better support for managing clones in software systems, we first developed *SimEclipse*: a clone-aware software development platform, and then, using the tool, we performed a study to investigate the usefulness of using a number clone based technologies in an integrated platform rather than using those discretely. Finally, a small scale user study is performed to evaluate *SimEclipse*'s effectiveness, usability and information management with respect to some pre-defined clone management activities. We believe that both researchers and developers would enjoy and utilize the benefits of using *SimEclipse* for different aspects of code clone research as well as for managing cloned code in software systems.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments—*integrated environments*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*restructuring, and reverse engineering*

General Terms

Design, Management

CASCON 2015, Markham, Canada.

Keywords

Software Clone, Integrated Clone Management, IDE Plugin

1. INTRODUCTION

Investigating and understanding a software system's code clones is important to manage clones easily and efficiently. Göde [7] found that most of the clones detected by state-of-the-art clone detectors need not be removed. His study has significant implications from the point of view of software maintenance. No matter what the cause, in most cases it is the developer's activity that directly (e.g., by copying, pasting or modifying code) or indirectly (e.g., by using code generation programs or tools) introduces cloned code in a system. It is very hard to manually track, especially in large systems, whether those activities unknowingly introduce clones. Without efficient access to clone information it is hard for a developer to efficiently manage clones in the system (e.g., refactoring/removing cloned code) or to reduce the negative effects of clones during software evolution. Even if information about the clones in a system is available (e.g., using a standalone clone detection tool), in most cases a developer needs to supplement that knowledge with manual investigation and identification on the working codebase to locate and process the clones. In addition, it is problematic to identify consistently changing clones over time [11].

In support of code clone management, a number of standalone tools are proposed in the literature and are available for use in clone detection, visualization, analysis, and so on [16, 17] . However, using these standalone tools for managing clones in evolving software might not always be useful in practice as opposed to having similar facilities directly available from within an IDE (Integrated Development Environment) as added features. Therefore, from a developer's perspective, there is a gap between state-of-the-art software clone management support tools and software development environments. Thus a clone tracking and awareness tool is essential for assisting the software developers to efficiently maintain software. A growing need for further research towards an integrated clone management system was also addressed in a recent study by Roy et al. [17].

Our overall idea for supporting clone management is to see if it is beneficial to integrate some key clone management primitives into a software development environment. That is, we are interested in addressing the following research question: *"Does integration of clone based technologies into developers working environment (IDE) make clone management activities more convenient and efficient (as opposed to using the those discretely, outside of IDE)?"*.

In order to answer this research question, we conducted a user study[1] based on a prototype IDE plugin *SimEclipse*[2] - an Eclipse[3] IDE plugin to help the developers for detecting, visualizing and tracking clones in projects from within IDE.In support of understanding the evolution of both clone and non-clone source code in a multi-version software system, *SimEclipse* also provides a source code evolution history viewer and clone genealogy viewer. We show how *SimEclipse* can support a software developer in dealing with clones and can seamlessly aligned with development workflow.

Overview: The rest of this paper is organized as follows. Section 2 covers some basics of clone management. Section 3 presents the features of *SimEclipse* plug-in. Section 4 and 5 provides the user study in detail based on *SimEclipse*. Section 6 covers some related work and Section 7 answers the research question. Finally, Section 8 concludes the paper.

2. CLONE MANAGEMENT

Clone management is a cross cutting topic concerning different domains of software clones. As an umbrella activity it covers various aspects of clones including but not limited to clone detection, classification, visualization, evolution analysis, tracking, refactoring, and so on. Current studies show various benefits of clone management, including improved customer satisfaction and improved system quality with a positive impact on software maintainability [15, 13]. In the remainder of this section we discuss some basic clone management features and clone management strategies.

2.1 Clone Management Features

- *Detection:* Clone detection is a fundamental requirement of a clone management strategy. If this feature is not present natively, detection results from third-party standalone clone detectors are used. Such feature dependency, or stated another way, lack of native detection support, would make a clone management system less interactive for the developers and also might be incapable of supporting other features like focused/on-demand clone searching, clone tracking, and so on.

- *Representation/Documentation:* This feature captures and stores the result of clone detection as a form of clone documentation that records the location of code segments and their clone relationships. Once created, it can be re-used as many times as needed to view the clone information for the system without re-invoking the detection process. Clone documentation may be analyzed to justify the use of clones or to find potential clones for removal.

- *Visualization:* Visualization is a powerful technique that can aid understanding and analysis of clones after detection. This feature may offer various types of clone visualization with (e.g., tree-view) or without (e.g., scatter-plot, tree-map) using internal features of the IDE. In most of the visualization approaches, the developers can navigate to the actual cloned code in the system and possibly compare the clones in the same group to view differences.

- *Analysis:* This module analyzes the clones in the system and provides various insights about the cloning status of the system. Such analysis may include but is not limited to various statistical analysis featuring types of clones, clone density in various regions in the system and historical analysis. In addition, analysis could be performed to determine clone candidates for refactoring with the purpose of removing duplication.

- *Tracking:* This module provides clone awareness functionality in a clone management system. An evolving software system goes through frequent changes (add / edit / delete) in the code base. Such changes may introduce new code segments that might form new clones (e.g., copy-paste code) or invalidate some existing clones in the system. Both situations require corresponding updates to the clone documentation, which can be supported with this feature. This feature tracks existing clones in the system and follows their evolution throughout the life of the software. In addition, it looks for source code changes and notifies the developers if new clones are introduced in the system because of the changes made.

- *Refactor/Removal:* This module allows the developers to delegate the code structure changes to the refactoring engine of the IDE and reduces errors that may occur if changing the code was done manually. This feature may provide guidance for refactoring, however, integrating this component into an IDE may still require that the clone information is provided manually to the refactoring engine.

2.2 Clone Management Strategies

Clone management summarizes all process activities which are targeted at detecting, avoiding and/or removing clones [6]. There are essentially two main strategies for managing clones:

1. *Preventive Clone Management* avoids introducing clones and may also maintain awareness of existing unavoidable clones to better understand their evolution in order to eliminate harmful effects. In practice, clones are unavoidable and thus the expectation of having a clone-free system is unrealistic. Therefore, preventive clone management might also be termed as *proactive* clone management [9] that aims to deal with the clones during their creation or soon after they are introduced.

2. *Corrective Clone Management* features refactoring/removing clones after they are detected. While it might not be possible to refactor/remove all of the clones in a system (e.g., Type-2[4] and Type-3[5] near-miss clones), exact clones (Type-1[6]) are typically good candidates for refactoring.

Next we present *SimEclipse* as a versatile IDE plug-in that covers a rich set of features in support of better clone management in software development and evolution.

[1]Study Material: http://homepage.usask.ca/ mdu535/tools /simeclipse/study
[2]Prototype: http://homepage.usask.ca/ mdu535/tools.html
[3]http://www.eclipse.org

[4]Code fragments which are structurally/syntactically similar but may contain variations in identifiers, literals, types, layouts and comments.
[5]Code fragments with modifications in addition to those defined for Type-2 clones, such as: insertion, deletion or update of a statement.
[6]Code fragments which are identical without considering the variations in white space and comments.

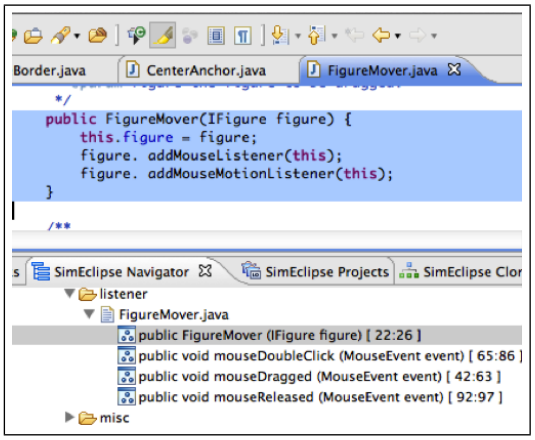


Figure 1: Source navigation in *SimEclipse Navigator View*

3. *SIMECLIPSE*: A CLONE-AWARE PLUG-IN FOR ECLIPSE IDE

Over the past decade, a number of state-of-the-art tools have emerged from software clone research that address clone detection, visualization, analysis and management. From the software development perspective, the question is, are those tools really useful in a practical context? That is, how conveniently can the software developers make use of these tools in their development activities. In most cases, the existing approaches partially cover the scope of clone management or do not integrate well in a software developer's development workflow. In this section, we present *SimEclipse* - an Eclipse plugin featuring the preventive clone management strategy by providing a developer options for detecting, visualizing, analyzing and tracking clones in projects. The goal of *SimEclipse* is to make state-of-the-art clone management technologies available in the IDE so that clones in software can be managed in a convenient way at the place where they are mostly introduced and evolved. Brief descriptions of the various *SimEclipse* features follow.

3.1 Just-in-time Clone Detection

Clone detection can be performed on a *SimEclipse* enabled project using various views, e.g., 'Eclipse Project/Package Explorer View', 'SimEclipse Projects View', 'SimEclipse Navigator View,' and so on. The 'SimEclipse Navigator View' (Fig. 1) also allows the developers to explore the codebase of a *SimEclipse* enabled project using a tree based hierarchical view. The developers can see the directory, files and fragments in the code base for which clone detection can be invoked for any item as per choice. Clicking on any file or fragment entry here will display the actual source file in the workbench using the integrated source editor associated with the file. For code fragments, the entire fragment will be highlighted in the editor (Fig. 1). In addition, a user can perform clone detection from the editor by defining an arbitrary portion of the code.

3.2 Clone Visualization

Detected clones are visualized through the 'SimEclipse Clones View' (Fig. 2) where clone groups are displayed based on the detection settings for the project. Clones are shown in different colors based on their types. Users can click on any clone to see it highlighted in the editor (Fig. 2). From this view, the user can also compare any two clones in a clone group to see the actual similarities and dissimilarities.

3.3 Clone Tracking

3.3.1 *Clone Annotation (Existence Tracking)*

The developers can have their editor marked for a file that contains clones by choosing the corresponding option in the context menu available by right-clicking on the source file in the editor. Each yellow flag mark shown in the left vertical bar of the editor (Fig. 3) represents the location of a clone fragment in the subject file. Location information of other clone fragments related to this clone group can be seen by double clicking the flag. This will open a pop-up window (Fig. 4) from which the developers can explore and see the actual source of those fragments in the editor.

3.3.2 *Clone Change Tracking*

Clone change tracking is an automated service provided by *SimEclipse* to track the generation and evolution of clones in the project. The service provides real-time notifications when any new clone has been evolved in the system due to the changes made in source code. If the changed code is originally a clone but the recent changes made on it is not quite enough (so that it will no longer be considered as clone under the defined detection setting), it will also be reported. Both the new and old (but changed) clone fragment entries will be highlighted in red text in the *SimEclipse Clones View*, while the others will remain in the default color scheme for the particular type of clone (Fig. 5). As a background service, it senses changes made on the codebase after every save operation made on the project source files from within the IDE and reports any cloning activity as happened. Using this feature, the developers can avoid accidental generation of clones, for example, because of not knowing the existence of similar code elsewhere in the project. If source code gets changed outside the IDE that affects the cloning state of the system, manual *update of the clone index* (a user operation in the plug-in) for that project would cover those changes in the clone tracking process. The current *SimEclipse* prototype does not provide any automatic refactoring or clone removal feature. Instead, it lets the developers do this based on their own rationale using a manual approach or using the code refactoring features available in the IDE. *SimEclipse* however does check whether changes made to any clone fragments made these disappear or not.

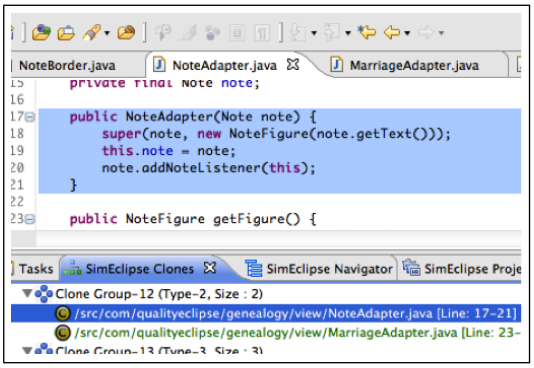


Figure 2: Inspect Clone Code in *SimEclipse Clones View*

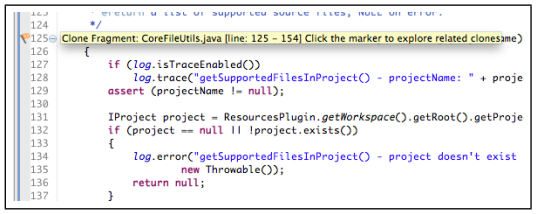


Figure 3: Marked location of clones in editor

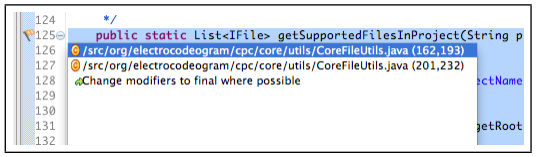


Figure 4: Marked Location shows other clones

3.4 Clone Analysis

3.4.1 *Clone Genealogy Viewer*

An implementation of the genealogy extractor proposed by Saha et al. [18] is integrated into the plug-in. For a multi-version project, *SimEclipse* is able to extract and display clone genealogies. From the *SimEclipse* settings option, users can add multiple versions of the project source if available (form the file system) and then invoke the *extract/view clone genealogy* operation for the lifecycle of the project as represented by the versions added. From the 'SimEclipse Clone Genealogy View' (Figure 6) users can explore the genealogies by versions and study their evolution.

3.4.2 *Code History Exploration*

Similar to the 'Clone Genealogy View', for a multi-version project, *SimEclipse* allows the developers to inspect the source modification history for both clone and non-clone source fragments (Figure 7). If the original project source code comes with source code repository information along with it, *SimEclipse* displays the developer/author information as well (using SVN/Git blame command internally) so that the developer who is currently working on the code would be able to know which other developers worked on this code in the past. This feature would help the developers to understand the changes in source code throughout the lifecycle of the project.

4. IDENTIFYING THE EFFECTIVENESS OF INTEGRATED CLONE TECHNOLOGIES: A SCENARIO BASED STUDY

The motivation of this study is to capture the user experience in dealing with software clones in two different working scenarios: the *Discrete Approach* and the *Integrated Approach*. For the *Discrete Approach*, the study participants

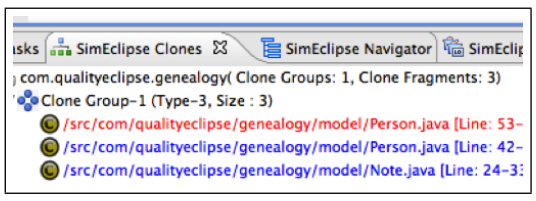


Figure 5: Notification of newly introduced clone

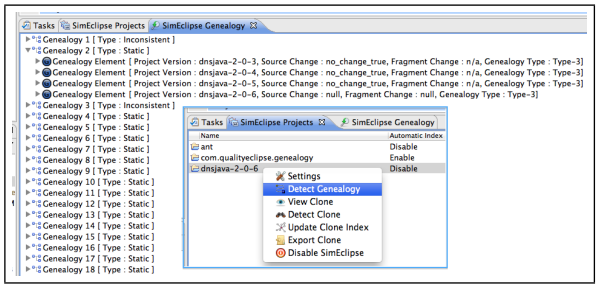


Figure 6: Clone Genealogy Viewer in *SimEclipse*

were told to perform some pre-defined code clone related tasks on a subject system using a set of standalone tools. For the *Integrated Approach*, they were told to perform the same tasks using *SimEclipse* as a tool with integrated clone management features. The idea is to capture whether they feel any necessity of having a platform with an integrated clone management system rather than discretely using some standalone tools to accomplish the tasks. The user experience captured in both approaches is contrasted to measure the effectiveness of using integrated clone technologies when working with software clones.

4.1 Experimental Setup

The study required a set of participants, a set of tasks, a questionnaire, methods of evaluation, and result analysis. The following steps constitute the setup for our experiment.

4.1.1 *Test Design*

In this experiment, the participants perform a set of six tasks in two different working scenarios/environments as stated earlier. The tasks were designed from easy to fairly complex covering different needs in clone management. Each task is associated with a question and the participants are expected to find some answers by performing the task. All the required input information to perform a task are also provided. The tasks designed for the study are as follows:

T1: Given three functions/methods (in the source code), identify if they are clones or not in the given system.
Target Usage: Clone Detection

T2: On-demand/Focused clone search, identify number of Type-1 clones in three source packages/folders.
Target Usage: Clone Detection/Visualization

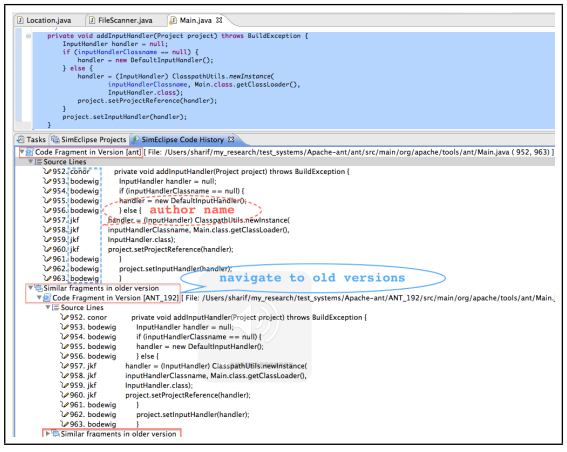


Figure 7: *SimEclipse* Code History Explorer

T3: Apply given source modification to three existing functions, if they are clones; apply the modification to all the associated clone fragments as well.
Target Usage: Clone Detection/Visualization/Tracking

T4: Add three functions on a specified source file (both code to write and source location are provided), determine which of the functions newly added fall into existing clones in the system, if so, identify the location and type of the clones.
Target Usage: Clone Detection/Visualization/Tracking

T5: Identify clone genealogy of two given functions (the function being a member of an existing clone class).
Target Usage: Clone Analysis/Understanding

T6: The clone genealogy of one of the above two functions is inconsistent. Identify that inconsistent genealogy and locate the change that makes the clone class inconsistent. Locate the developer who made the change.
Target Usage: Clone Analysis/Understanding

Finally, a questionnaire was designed to evaluate the tasks performed by the participants in terms of *Completion Status*, *Completion Time*, *Correctness* and *Execution Difficulty*.

4.1.2 Running Study Session

This part of our experiment modeling initiates the user participation in a one-on-one session with each participant. At the very beginning, the participants were given an introduction to the nature and purpose of the study followed by a small orientation on software clones if the term is new to the participant. After that, a short survey was performed to capture their familiarity with software development and various aspects of clones in software. The next part allows the participants to do specific tasks using some code clone related software tools and answer the related questions. This part of the study session had the following three phases:

Phase 1 (*Discrete Approach*):

1. A candidate Java project with multiple versions (at the release level) is identified. Each version is checked out from the SVN source repository to separate folders with its version name.
2. The latest version of the candidate project is imported into the Eclipse IDE as a Java project.
3. Standalone clone detection, visualization, analysis tools are provided (we used *SimCad*, *VisCad*, *gCad*) as well as the SVN command-line client to explore the author information from the source code repository.

   Study Session: The study participants are shown the use of the supporting standalone tools (using video demonstration) and then asked to perform the tasks mentioned earlier in this section. At the end of each task, the participant fills out the 'Task Based Questionnaires' for this task on the *Discrete Approach*, which records his/her experience and any outcome of the task in this scenario.

Phase 2 (*Integrated Approach*):

1. A candidate Java project with multiple versions (at the release level) is identified. Each version is checked out from the SVN source repository to separate folders with its version name.
2. The latest version of the candidate project is imported into the Eclipse IDE as a Java project
3. The *SimEclipse* plugin is installed and enabled for the candidate project that was imported into Eclipse.

   Study Session: The study participants (the same group in the previous study) are shown the features of *SimEclipse* (using video demonstration) and then they are asked to perform the same tasks (with a different input set) again using the various features of *SimEclipse*. At the end of each task, the participant fills out the 'Task Based Questionnaires' for this task on the *Integrated Approach*, which records his/her experience and any outcome of the task in this scenario.

Phase 3:

The participants fill out the answers to the questions for the "Overall User Experience" in performing the tasks in two different environments.

4.2 Summary of Findings

There were 12 participants in the study. All of them with some software development experience throughout their academic curriculum, while five of them with at least one year of experience in software industry. Among the participants, eight were found having at least basic knowledge of software clones prior to the introductory session of the study. Since, in our study, a participant gets to perform similar tasks in two test scenarios (with different input though, clone set of similar size in different project), the order of the test scenario might introduce some bias in test results. To avoid this, we had 50% of the participants to try the *Discrete Approach* first and then the *Integrated Approach*, and in the reverse order for the remaining 50%. User feedback from the participants upon performing the given tasks in the two different working scenarios has been analyzed. The result is presented as follows in the form of contrasting their task performance based on four criteria: completeness, time taken, correctness and difficulty.

Task Completeness Analysis: We compare how many tasks the participants in the two different study environments completed successfully. Figure 8 shows, in the *Discrete Approach*, three and seven participants were not able to complete the task *T4* and *T6* respectively in the given timeframe (15 minutes/task). On the other hand, all the participants in the *Integrated Approach* were able to complete all the tasks successfully by the given time.

Task Completion Time Analysis: In the task completion time analysis, we analyze and compare the time taken by the participants in completing the tasks in the different environments. For the ease of time performance comparison, the given timeframe (15 minutes/task) is divided into three equal time slots labeled as Fast, Medium and Slow. The recorded time slot data in the study are normalized into *Time Points* according to the Table 1. The normalization

Table 1: Point scale for task completion time analysis

Completion Time Slot	Time Point
0-5 minutes / Fast	3
5-10 minutes / Medium	2
10+ minutes / Slow	1

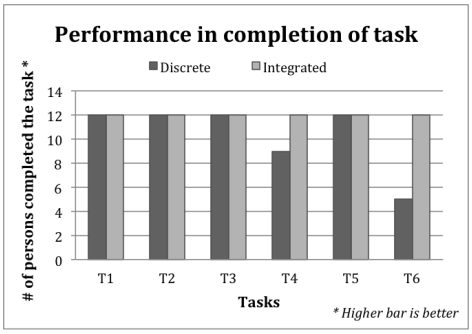


Figure 8: User performance in completing tasks in the two different environments

Table 2: Point scale for task correctness analysis

Task Based Question Answered	Correctness Point
Wrong/Incomplete	0
Partially correct	1
Fully correct	2

scheme assigns higher points to the task completed in less time. For each task, the total and average *Time Points* are calculated for all the participants. Figure 9 shows the comparison of the total and average *Time Points* of the tasks in the two different environments. From the figure it is clear that all the tasks being performed in the *Integrated Approach* associated with higher total and average *Time Points* than in the *Discrete Approach*, which means the participants were able to complete the tasks faster in the *Integrated Approach* compared to the *Discrete Approach*.

Task Correctness Analysis: Each task has a question associated with it, and by performing the task, a participant is lead towards an answer. This analysis covers user performance in finding the correct answer to the question associated with the tasks. Based on the participant's answer, a *Correctness Point* is assigned according to Table 2. Note that the participant's answer to some tasks might be considered as partially correct when there are multiple components in the answer. For example, correctly identifying some of the clones of a given code fragment, but not all. For each task, total and average *Correctness Points* are calculated for all the participants. Figure 10 shows the comparison of total and average *Correctness Points* for the tasks in the two different setups, which clearly indicates that the participants did well in the *Integrated Approach* compared with

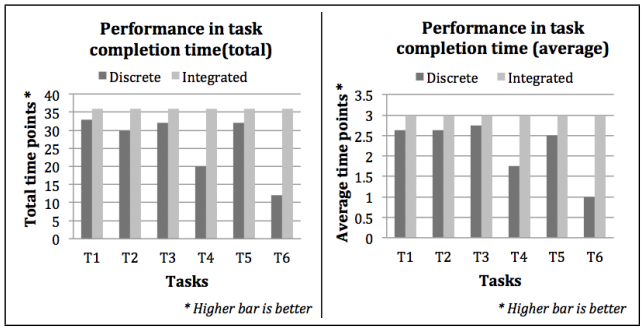


Figure 9: Comparison on completion time of tasks taken by users in different scenario

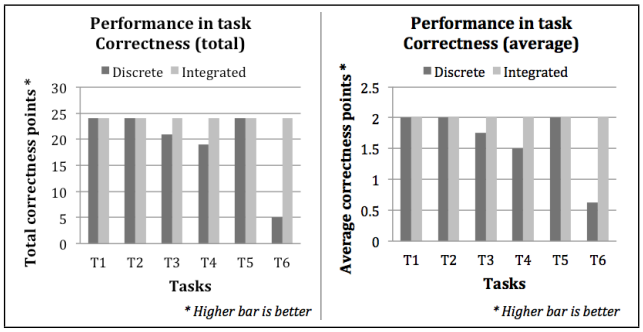


Figure 10: Comparison of task correctness in the two different environments

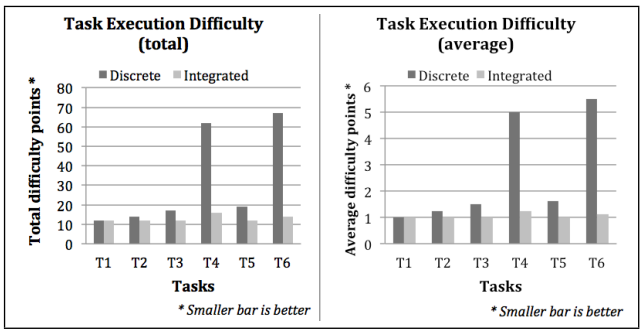


Figure 11: Comparison on difficulty of performing the tasks by the users in different scenario

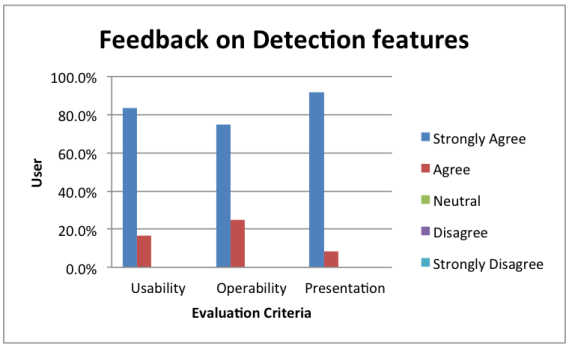
the *Discrete Approach* when finding correct answers to the questions.

Task Difficulty Analysis: This part of the analysis investigated how difficult it was to perform the given tasks using the resources provided in each scenario. The participants reported their opinions in a six level difficulty scale, where each level is assigned a *Difficulty Point* according to Table 3. For each task, total and average *Difficulty Points* are calculated for all the participants. Figure 11 shows the comparison of total and average *Difficulty Points* for the tasks performed by the participants in the two different environments, which again shows that the participants performing the tasks in the *Integrated Approach* felt that the tasks were less difficult to complete than in the *Discrete Approach*.

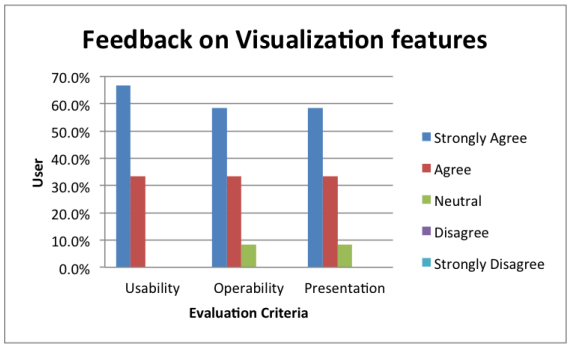
Finally, from the feedback received on *Overall user experience question*, 100% of the participants felt more comfortable and confident in performing the given tasks in the *Integrated Approach* than in *Discrete Approach*. Moreover, eight (66%) participants mentioned 'IDE' as a preferred platform over a 'Standalone Application Suite' for providing tool support for clone management. Some major reasons in support to their opinion were: avoidance of context switching between IDE and some external tool to transfer clone knowl-

Table 3: Point scale for task difficulty analysis

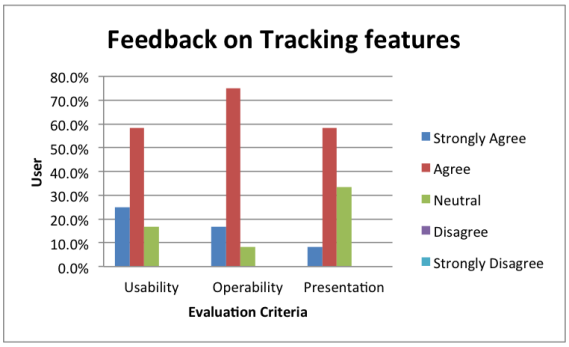
Task Difficulty Level	Difficulty Point
Very easy	1
Easy	2
Somewhat easy	3
Somewhat hard	4
Hard	5
Very Hard	6



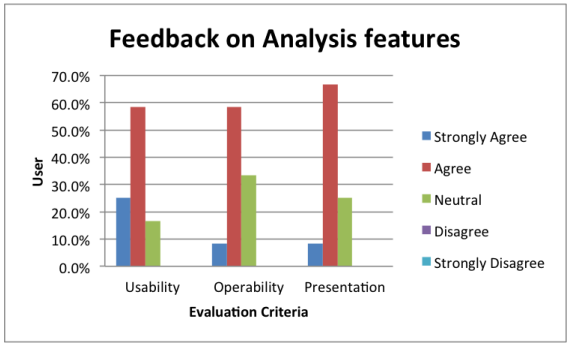
(a) Detection Feedback



(b) Visualization Feedback



(c) Analysis Feedback



(d) Tracking Feedback

Figure 12: User opinion on Usability, Operability and Presentation for the features of *SimEclipse*

edge, less manual or intermediate data processing and reducing the possibility human error. Among the rest, two (17%) participants were fine with either approaches, and the remaining two (17%) preferred *Standalone Application Suite* over IDE with supporting reasons including: possibility of IDE being bogged down with memory/time intensive tasks (specially for large systems), freedom for the developers to choose working with different IDEs, etc. Therefore, based on the feedback from the majority of the participants, we can recommend that an IDE integrated clone management tool could make the clone management tasks easier, less error prone and leads to faster processing.

The table 4 shows task-wise t-test [8] analysis to verify whether the difference of performance of the participants in the two test environments is significant or not. From the table it appears that, participants would be able to do

Table 4: Significance of differences in task performance by the study participants

Task	Completion Status	Completion Time	Correctness	Execution Difficulty
T1	❍	◗	❍	❍
T2	❍	●	❍	◗
T3	❍	●	◗	●
T4	◗	●	●	●
T5	❍	●	❍	●
T6	●	●	●	●

● Significant Difference ($P \leq 0.05$)
◗ Insignificant Difference ($P > 0.05$)
❍ No Difference

various clone management tasks faster and easier way in an integrated environment. It also shows, when the complexity of the task increases, the participants tend to perform the clone management tasks better in integrated approach.

4.3 Threats to Validity of the Experiment

- In the 'Task Correctness' measurement, the state of answer termed as 'partial correct' has not been weighted (for task correctness points) as per the number of the correct answers. Therefore, there would be no difference whether the (multi-part) answers for a task found by the participants were almost correct or almost incorrect. Considering this would make the comparison more realistic.
- The scales we used for 'Time Point' and 'Difficulty Point' are defined arbitrarily, but with a particular order and uniform difference between values. Since, we are contrasting performance values (sum/average) for the same task (in two different test scenario), we believe these scales would acceptably be able to reflect the task performance difference.
- The number of participants in our study is not sufficient enough to draw a general conclusion. Furthermore, the level of technical expertise of the involved participants might also have some effects on the experimental results.

5. SCENARIO BASED FEATURE EVALUATION OF *SIMECLIPSE*

In this section we describe how we evaluated the usefulness of *SimEclipse* in practice for clone management. This

is an extension of the previous study that covers an evaluation of the outcomes of the tool by the participant developers. Since the correctness evaluation of the core technologies behind *SimEclipse* (e.g., clone detector [20], genealogy detector [18], etc.) have been considered in their respective studies, we did not conduct a similar evaluation of *SimEclipse* here. Instead, we focus on investigating its usefulness and acceptability to its target users.

5.1 Study Design

The study was based on the following criteria, measured from the user's task based experience of using the tool:

> *Usability:* How effective the tool is to manage clones in a system during software development?
>
> *Operability:* How simple and easy the tool is to operate for source code clone management?
>
> *Presentation:* How presentable the information is in the tool for the users in understanding and analyzing clones?

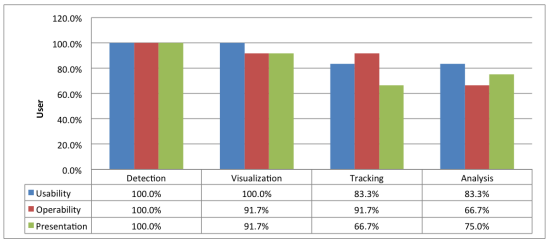
A set of tasks was designed to cover all the features of *SimEclipse*. After finishing the tasks, the participants were required to answer some questions addressing *Feature Evaluation*, *Overall Tool Experience* and *Issues and Improvements* of *SimEclipse* as an integrated clone management tool.

5.2 Summary of Findings

5.2.1 Feature Evaluation

This section presents an analytical overview of the user evaluation of different features of *SimEclipse* in managing clones in software. User feedback on the feature evaluation questions are analyzed and presented in Fig. 12 with a separate component for each feature. From the feedback chart, it is clear that the features of *SimEclipse* got most of the remarks on the positive side of the Likert scale from the study participants on all three evaluation criteria (*Usability, Operability* and *Presentation*). Figure 13 presents a consolidated view of the positive feedback ('Strongly Agree' or 'Agree') was received for all the features of *SimEclipse*. The observation shows, 100% of the participants expressed their positive feedback (answering 'Strongly Agree' or 'Agree') on all the evaluation criteria for the *Detection*, and for *Visualization* features it was found to be 91.7%. For the remaining two features, at least 66.7% of them were positive in the three evaluation criteria for *Analysis* and *Tracking* features. This indicates the *Analysis* and *Tracking* features need to be improved to further enhance the clone management experience using *SimEclipse*. The study also received various recommendations from users in improving and adding new features to *SimEclipse*, which will be discussed in the following section.

Feedback from the *Overall Experience Questions* has been analyzed and presented in Figure 14. Among the study participants, 83.3% of them agreed that *SimEclipse* would be *Helpful in Clone Management*, while 66.7% of them agreed that *SimEclipse* had *Overall Good Performance*, and 75% were found to be *Overall Satisfied* in using the tool.



	Detection	Visualization	Tracking	Analysis
Usability	100.0%	100.0%	83.3%	83.3%
Operability	100.0%	91.7%	91.7%	66.7%
Presentation	100.0%	91.7%	66.7%	75.0%

Figure 13: Consolidated feedback on *SimEclipse*'s features

5.2.2 Future Improvements

At the end of the study session, the *Issues and Improvements Questions* enabled us to obtain advice on how to improve the overall clone management experience of the developers using *SimEclipse*. We received valuable suggestions on improving existing features, especially for the interactive visualization user interface part of Clone Tracking and Clone Analysis. Although *SimEclipse* is capable of performing a focused clone search, a few users requested post-detection filtering of clones when detection is done on the whole project, and a search and filtering option in the *Clone Genealogy View*. To enhance the clone visualization experience, the implementation of a model/graph based visualization also was suggested by some participants. For additional features, notable requests were: enhanced highlighting of differences in clone comparisons, portable clone documentation sharable among the developers, clone refactor scheduling, and linked-editing of clones. We believe, these would be some great addition to our future improvement plan to enhance the overall clone management experience using *SimEclipse*.

6. EXISTING WORK ON IDE BASED CLONE MANAGEMENT

There are many clone detection tools; each has its own strengths and weaknesses. However, for proactive clone management, the support for clone detection should be integrated with the development process. Some of the IDE based tools are mentioned here categorically as follows.

Tools for managing copy-paste clones: This category covers tools that only deal with clones resulting from a developer's copy-paste operation. Hou et al. developed their clone management tool CnP [9] so that it is tightly coupled with a clone detection technique based on a programmer's copy-paste operations. Thus, the scope of clone management is limited to copy-pasted code only, and not applicable to clone management based on similarity based clone detection. Jablonski and Hou introduced CReN [10] tool as

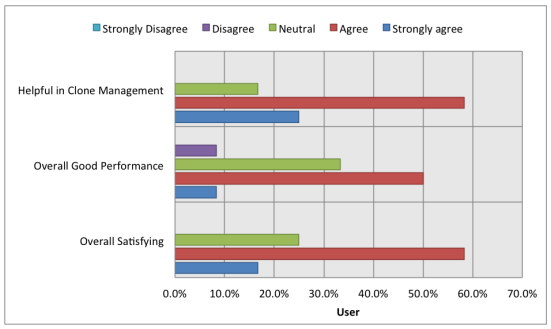


Figure 14: Overall user evaluation of *SimEclipse*

an Eclipse plug-in for Java programs to help programmers avoid making copy-paste error, while renaming various instances of identifiers, such as variable names. It tracks the code clones involved when copying and pasting occurs in the IDE and infers a set of rules based on the relationships between the identifiers in these code fragments. So, it is only a subset of Type-1 clones (i.e., only the clones arising from the developer's copy-paste action in IDE) that are being managed using this tool. The plug-in CSeR (Code Segment Reuse) was developed by Jacob et al. [12] to check copy-paste induced clones in an integrated development environment. The tool was designed to compute the clone differences interactively by checking whether a piece of code is copy-pasted while the programmer edits and/or types the code. A similar IDE plugin is proposed by Venkatasubramanyam et al. [21] for proactive moderation of the genesis of clones through copy-paste-modify operations. The approach is guided by associating constraints formulated from predefined guidelines, and checking for their satisfaction at the time of copy and upon modifications. CloneBoard [4] and CPC [22] are two Eclipse plugins that can detect and track clones based on clip-board/copy-paste activities of the developer. Both CPC and CloneBoard support linked editing of clone pairs. While the above mentioned approaches based on a programmer's copy-paste activities may be able to handle intentional clones (i.e., a clone that someone purposely introduced or is about to introduce in a system), they are unable to deal with unintentional clones. Moreover, such tools may not be suitable for distributed development, as they may fail to combine information about clones separately created by the developers working in a distributed environment [23].

Tools with integrated clone detection and visualization only: This second category of tools provides native clone detection support and offers various visualizations of clones from within the IDE. SHINOBI [14] is an add-on to the Microsoft Visual Studio 2005 based on client-server architecture. It relocates the clone detection effort from the client side (local programming environment) to a central server (source code repository). SHINOBI does not offer that much support for clone management except that displaying clones of a code fragment underneath the mouse-cursor. Besides, since it internally uses CCFinderX's preprocessor, it can detect Type-1 and Type-2 clones only [23]. AST-based clone detector CloneDR [2] is also available as an IDE plug-in that can detect Type-1 and Type-2 clones. However, it seems to fail in detecting Type-3 clones in many scenarios [16]. Zibran and Roy [23] presented an IDE-integrated clone search tool for Type-1, Type-2 and Type-3 clones. The implementation is based on a suffix-tree based k-difference hybrid algorithm. Bahtiyar developed JClone [1] as a plugin to the Eclipse IDE for detecting Type-1 and Type-2 clones only from Java projects. JClone applies an AST based technique to detect clones. It enables the user to trigger the detection of clones from one or more selected files or directories. It also offers a few visualizations (i.e., TreeMap and CloneGraph views). Plug-ins in this category may aid clone analysis to some extent, but they offer no further support for clone management beyond the detection and visualization of clones.

Versatile clone management tools: This final category of tools covers additional clone management services beyond those mentioned in the previous two groups. Duala-Ekoko et al. developed CloneTracker [5], a more versatile clone management system, as an Eclipse plug-in. The plug-in allows tracking and simultaneous editing of clones. It relies on the output of the SimScan [3] clone detection tool and requires the programmer to manually select the clone groups of interest to be documented. Recently, Tairas and Gray developed CeDAR [19], a plug-in for the Eclipse IDE, where they introduced an approach to unify the processes of clone detection, analysis, and refactoring. They integrated a clone detection approach in the plug-in and presented a visualization technique in which one clone instance displays the properties of all the clones in the clone group. This representation helps in refactoring as clone group representation displays the differences among clone instances.

Table 5: Comparison of *SimEclipse* with other tools

Features	CnP [9]	CSeR [12]	CloneBoard [4]	SHINOBI [14]	CloneDR [2]	Zibran and Roy [23]	JClone [1]	CloneTracker [5]	CeDAR [19]	SimEclipse
Integrated Detection Engine	○	○	○	●	●	●	●	○*	○*	●
Copy-paste Detection Only	●	●	●	○	○	○	○	○	○	●
Type-1 Clone Detection	◗	●	◗	●	●	●	●	●	●	●
Type-2 Clone Detection	◗	○	◗	●	●	●	●	●	●	●
Type-3 Clone Detection	◗	○	◗	○	◗	●	○	●	●	●
On-the-fly/Focused Detection	○	○	○	○	◗	●	◗	○	○	●
Clone Visualization (editor)**	◗	◗	◗	◗	◗	◗	◗	●	◗	●
Clone Visualization (model)	○	○	○	○	○	○	●	○	○	○
Code Change History View	○	○	○	○	○	○	○	○	○	●
Clone Genealogy View	○	○	○	○	○	○	○	○	○	●
Clone Tracking	○	○	●	○	○	○	○	●	○	●
Clone Refactoring	◗	○	◗	○	◗	○	○	○	●	○

● Full Support ◗ Partial Support ○ No Support
* Use clone detection results from other standalone tool
** Display clone class, highlight, compare clones in editor etc.

Our integrated clone management plug-in *SimEclipse* has been developed as a versatile approach to deal with clones by making a number of clone based technologies readily available to the software developers and clone researchers. The goal is to allow the developers to detect, understand and manage clones in software they are developing both in time and effort efficient ways. From the feature comparison (only covers feature availability, not its strength/weakness) as shown in Table 5 with some of the existing clone management tools, it is clear that *SimEclipse* supersedes the rest of the tools in providing integrated features to identify and understand software clones as well as address various clone management needs.

7. ADDRESSING THE RESEARCH QUESTIONS

The research question in this study was about investigating the effectiveness of integrating clone based technologies into a single platform. The study described in Section 4 provides some evidence for answering this question. The participants in the study were allowed to perform some clone based tasks using discrete clone support tools and then using an IDE integrated clone management plugin. Based on their feedback after performing the same tasks in the two different

environments, it is clear that having various clone management functionalities under a single platform yields faster, convenient and less error prone clone management. Again, the feedback gained from the 'Overall user experience questions' at the end of the study described in Section 4 shows 100% of the participants felt more comfortable and confident in performing the given tasks using an integrated tool in an IDE rather than using some discrete standalone tool.

In support of the study, we developed *SimEclipse*, a clone-aware IDE plugin to provide the software developers a platform for managing clones from within an IDE. In the second part of our study described in Section 5, we evaluated *SimEclipse* based on three evaluation criteria (*Usability, Operability and Presentation*) of its features. We found, 83.3% of the participants considered *SimEclipse* would be *Helpful in Clone Management*, while 66.7% of the participants agreed that *SimEclipse* had *Overall Good Performance*, and 75% were *Overall Satisfied* in using the tool.

8. CONCLUSION AND FUTURE WORK

Lack of integration of clone management functionality in a developer's work environment makes the task of managing clones in software a challenging and potentially error prone task. Our study here does not focus so much on presenting a better technique for finding clones, but on an infrastructure which makes the application of clone detection in a software engineering pipeline easier to use, thus more effective for its stakeholders. In this paper, we presented an empirical study in order to find a way to provide efficient clone management support to the developers. Here, we presented a clone-aware software development platform *SimEclipse* as a handy tool to accomplish the task of clone detection and various clone management activities in an easier and meaningful way. The objective of developing such a tool was to make the clone management activities as a part of the software development lifecycle. As clones in software are mainly evolved from various coding activities of the developer, we have shown that ready-made tool support for clones in a typical software development platform (e.g., IDE) as a plugin would make a great impact in getting efficient control of managing evolution of clones in software systems. Our small scale user study shows that *SimEclipse* has great potential to be used both in research and industry in dealing with clones.

In the future, we would like to focus mainly on improving *SimEclipse* by addressing the feedback and improvement suggestions we received from our tool evaluation study, especially the integration of clone refactoring capability. The ultimate goal will be to enhance the end-to-end clone management experience of the developers using *SimEclipse* and thus promoting the practical use of code clone research.

9. REFERENCES

[1] M. Bahtiyar. JClone : Syntax tree based clone detection for java. Master's thesis, Linnaeus University, 2010.

[2] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *ICSM*, page 368, 1998.

[3] B. E. Bulgaria. *SimScan - Similarity Scanner*. http://blue-edge.bg/download.html, last access: Aug 2011.

[4] M. de Wit. *Managing Clones Using Dynamic Change Tracking and Resolution*. M.Sc. thesis, Delft University of Technology, 2008.

[5] E. Duala-Ekoko and M. Robillard. CloneTracker: tool support for code clone management. In *ICSE*, pages 843–846, 2008.

[6] S. Giesecke. Generic modelling of code clones. In *DRSS*, pages 1–23, 2007.

[7] N. Göde. Clone removal: Fact or fiction. In *IWSC*, pages 22–40, Cape Town, SA, 2010.

[8] F. Gravetter and L. Wallnau. *Essentials of Statistics for the Behavioral Sciences*, volume 8th ed. 2013.

[9] D. Hou, P. Jablonski, and F. Jacob. CnP: Towards an environment for the proactive management of copy-and-paste programming. In *ICPC*, pages 238–242, 2009.

[10] P. Jablonski and D. Hou. CReN: a tool for tracking copy-and-paste code clones and renaming identifiers consistently in the IDE. In *ETX*, pages 16–20, 2007.

[11] P. Jablonski and D. Hou. Aiding software maintenance with copy and paste clone awareness. In *ICPC*, pages 170–179, 2010.

[12] F. Jacob, D. Hou, and P. Jablonski. Actively comparing clones inside the code editor. In *IWSC*, pages 9–16. ACM, 2010.

[13] C. Kapser and M. Godfrey. Cloning considered harmful" considered harmful. In *WCRE*, pages 19–28, 2006.

[14] S. Kawaguchi, T. Yamashina, H. Uwano, K. Fushida, Y. Kamei, M. Nagura, and H. Iida. SHINOBI: A tool for automatic code clone detection in the IDE. In *WCRE*, pages 313–314, 2009.

[15] B. Lague, D. Proulx, J. Mayrand, E. Merlo, and J. Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In *ICSM*, pages 314–321. IEEE Computer Society, 1997.

[16] C. Roy, J. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Sci. Comput. Program.*, 74:470–495, 2009.

[17] C. Roy, M. Zibran, and R. Koschke. The vision of software clone management: Past, present and future. In *SEW*, page 16. IEEE, 2014.

[18] R. Saha, C. Roy, and K. Schneider. An automatic framework for extracting and classifying near-miss clone genealogies. In *ICSM*, pages 293 –302, 2011.

[19] R. Tairas and J. Gray. Increasing clone maintenance support by unifying clone detection and refactoring. In *IST*, volume 54-12, pages 1297–1307, 2012.

[20] M. Uddin, C. Roy, K. Schneider, and A. Hindle. On the effectiveness of simhash for detecting near-miss clones in large scale software systems. In *WCRE*, pages 13 –22, 2011.

[21] R. Venkatasubramanyam, H. Singh, and K. Ravikanth. A method for proactive moderation of code clones in ides. In *IWSC*, pages 62–66, 2012.

[22] V. Weckerle. CPC: an eclipse framework for automated clone life cycle tracking and update anomaly detection. Master's thesis, Freie Universität Berlin, Germany, 2008.

[23] M. Zibran and C. Roy. IDE-based real-time focused search for near-miss clones. In *ACM-SAC (SE Track)*, pages 1–8, 2012.