Effects of Cloned Code on Software Maintainability: A Replicated Developer Study

Debarshi Chatterji, Jeffrey C. Carver, Nicholas A. Kraft
Department of Computer Science
The University of Alabama
Tuscaloosa, AL, USA
dchatterji@ua.edu; {carver, nkraft}@cs.ua.edu

Jan Harder
Software Engineering Group
University of Bremen
Bremen, Germany
harder@informatik.uni-bremen.de

***Abstract*—Code clones are a common occurrence in most software systems. Their presence is believed to have an effect on the maintenance process. Although these effects have been previously studied, there is not yet a conclusive result. This paper describes an extended replication of a controlled experiment (i.e. a strict replication with an additional task) that analyzes the effects of cloned bugs (i.e. bugs in cloned code) on the program comprehension of programmers. In the strict replication portion, the study participants attempted to isolate and fix two types of bugs, cloned and non-cloned, in one of two small systems. In the extension of the original study, we provided the participants with a clone report describing the location of all cloned code in the other system and asked them to again isolate and fix cloned and non-cloned bugs. The results of the original study showed that cloned bugs were not significantly more difficult to maintain than non-cloned bugs. Conversely, the results of the replication showed that it was significantly more difficult to correctly fix a cloned bug than a non-cloned bug. But, there was no significant difference in the amount of time required to fix a cloned bug vs. a non-cloned bug. Finally, the results of the study extension showed that programmers performed significantly better when given clone information than without clone information.**

***Index Terms*—code clones; software clones; empirical studies; clone management; software maintenance; developer behavior**

I. Introduction

Developers often reuse code via copy and paste [13]. The resulting copies are known as code clones. While code clones are not inherently harmful [5], [6], [12], [20], [23], they are associated with long-term risks to software quality and maintainability. Relative to quality, if buggy code is cloned, the bugs are also cloned. Relative to maintainability, bug fixes and other changes must be propagated to all cloned fragments potentially resulting in increased maintenance effort. In addition, explicit or implicit links among cloned code fragments must be maintained when those fragments need to remain consistent.

Researchers have used retrospective analysis of software repositories to study the effects of code clones on quality and maintainability [2], [3], [8], [10], [15], [16], [18], [19], [23]. These *post hoc* studies have focused on qualities such as genealogical traits [3], [23] and clone propagation [15]. While *post hoc* studies can reveal important information, they do have limitations. For example, these studies examine snapshots of the source code, but are not able to understand the actions that occurred in between the snapshots. Hence, these studies may miss valuable information that could provide insight into the maintenance process. Therefore, to understand the effects of code clones on software maintainability, there is a need to actively observe developers as they maintain code. There is currently a dearth of this type of human-based empirical studies to complement the retrospective studies of code repositories.

In two recent developer studies, we investigated the effects of cloned code on bug localization [4] and on bug removal [7]. While the results of these studies provided insights into the behavior, efficiency, and effectiveness of developers tasked with understanding and removing bugs in cloned code, each study had limitations that needed to be addressed by additional studies. In the bug localization study [4], developers used a clone report to help them locate bugs in cloned code, but did not actually repair the bugs. The results of this study showed that proper use of clone information was helpful. In the bug removal study [7], developers were asked to fix bugs in cloned code, but were not provided with clone information. This study did not find any significant difference in the difficulty of fixing cloned bugs compared with non-cloned bugs.

To better understand the factors that may have affected the results in these prior studies, to provide additional evidence regarding the effects of code clones on software maintainability, and to understand the impact of providing developers with clone information during a debugging task, we designed a replication of the bug removal study by Harder and Tiarks [7]. Replication of experiments in different environments with different populations is necessary for validation and generalization of results [1], [11], [21]. Although the level of involvement of the original experimenters in the replication is a debated topic [11], [14], [17], [22], we opted to collaborate with Harder to ensure the fidelity of the replication and to draw the most benefit from his experiences conducting the original study. During the replication, Harder acted as a consultant, helping in the setup of the study design and the data analysis. He shared the laboratory package used for the original study which consisted of the archives of the Eclipse plugin used for this study and the source code for the software systems along with an instruction manual.

The remainder of the paper is organized as follows. Section II describes the study design. Section III reports the results. Section IV compares the results of the original study and the extended replication. Section V identifies threats to validity. Section VI summarizes the contributions of the paper and describes future directions.

II. Study Design

The overall goal of this study was to investigate the effects of cloned code on software maintainability via a replicated experiment. In this case, the replication was an *extended replication*, which has two parts:

- The *Replication* - a strict replication of the original study [7] to confirm (or refute) the results of the original study, and
- The *Extension* - an extension of the replication to determine whether developers were more effective in removing bugs when provided with code clone information.

Harder and Tiarks developed a laboratory package for the original study that investigated the effects of cloned code on the performance of developers performing debugging tasks [7]. We used this laboratory package to conduct the *Replication* and an augmented version of it (which included clone information about the subject software systems) to conduct the *Extension*. We used the same study design for both the *Replication* and the *Extension*, with the exception of providing the participants with clone information during the *Extension*. Compared to the original study design of Harder and Tiarks, the design of the replication introduced a few minor changes.

First, in the original study, each participant performed the study tasks during one of several sessions along with a small number of other participants, whereas in the replication, each participant performed the study tasks during two consecutive, 75-minute class meetings along with his/her classmates. Second, in the replication, each task was treated as a classroom assignment, and participants were given credit for attempting the tasks (rather than for completing the tasks successfully). Students in three distinct courses participated in the study.

A. Research Objectives

The study addressed three research questions:

RQ1: Does the time needed for a bug removal increase when the bug is cloned?

RQ2: Does the probability of incorrect bug removals increase when the bug is cloned?

RQ3: Does the performance of participants improve when they are provided with clone related information?

The *Replication* addressed *RQ1* and *RQ2*, which we adopted from the original study. The *Extension* addressed *RQ3*. In particular, we formulated and tested three null hypotheses, each of which is motivated by one of the research questions:

H_0^{time} The time needed for removal of a cloned bug is *less than or equal to* the time needed for removal of a non-cloned bug.

H_0^{corr} The probability of a correct removal of a cloned bug is *greater than or equal to* the probability of a correct removal of a non-cloned bug.

H_0^{cinf} The probability of a correct removal of a bug without clone information provided is *greater than or equal to* the probability of a correct removal of a bug with clone information provided.

The corresponding alternative hypotheses are:

H_A^{time} The time needed for removal of a cloned bug is *greater than* the time needed for removal of a non-cloned bug.

H_A^{corr} The probability of a correct removal of a cloned bug is *less than* the probability of a correct removal of a non-cloned bug.

H_A^{cinf} The probability of a correct removal of a bug without clone information provided is *less than* the probability of a correct removal of a bug with clone information provided.

B. Variables

The variables are the same as in the original study.

1) Independent Variables: (a) Programs — We used two software systems for the maintenance tasks: Frozen Bubble and Pacman (described in more detail in Section II-D). (b) Versions — Each software system had two different versions: one containing a multiple-instance (cloned) bug and one containing a single-instance (non-cloned) bug.

2) Dependent Variables: (a) Time — Seconds needed by a participant to finish the tasks. (b) Correctness — Three levels describing the removal of a bug:

Addressed: For a cloned bug, the participant removed at least one, but not all, of the instances of the bug.

Complete: The participant removed all instances of the bug. (Note that for the non-cloned bugs, this level is equivalent to the **Addressed** level)

Incomplete: The participant was unable to remove any instances of a bug, resulting in his or her data being excluded from the study.

C. Subject Selection

In the original study, the participants included 21 computer science students from the University of Bremen and 12 attendees of Dagstuhl Seminar 12071, which hosted researchers from the code clone community. All student participants were required to have taken a Java course before participating in the study. The original experimenters differentiated between novice and expert programmers. The student participants were deemed to be novice programmers. The Dagstuhl participants were deemed to be expert programmers.

In the replication, the 47 participants included 23 students enrolled in a junior-level software engineering course, 8 students enrolled in a senior-level software engineering course, and 16 students enrolled in a graduate-level software engineering course (all at the University of Alabama). All participants had previously taken at least one programming course. Similar to the original study, we assumed that we could split the participants into a novice group (undergraduate students) and an expert group (graduate students). In reality it was not possible to make this distinction based on the data collected from the background surveys. The participants' mean self-evaluation of Java familiarity on a scale of 1 (not familiar) to 100 (expert) was 42.96 for the undergraduates and 48 for the graduates. The difference of 5.04 between the groups was not large enough to treat them as separate groups as was done in the original study with the students and the Dagshtul attendees. The level of expertise of the participants in the replication is lower than that of the participants in the original study (i.e. in which participants not familiar with Java were excluded). Given the small sizes of the software systems and the low complexity of the bug fixes, we did not consider lack of familiarity with Java to be a threat to validity.

Another difference between the replication participants and the original participants is that only 2/3 of the replication participants were familiar with Eclipse while all of the original participants were familar with Eclipse.

All participants took part in both the *Replication* and the *Extension*. After collecting the data, we excluded 10 participants from the *Replication* and the *Extension* along with 2 additional participants from the *Extension* only due to one of the following reasons:

- Corrupt XML files in the workspace,
- Corrupt Eclipse log files (due to improper shutdown), or
- Incomplete tasks

Thus, we analyzed data for 37 participants in the *Replication* and 35 in the *Extension*.

D. Instrumentation

This section describes the software systems and the plug-ins that the participants used to perform the tasks.

1) Software Systems: FrozenBubble (FB) and Pacman (PC) are both open source games with fewer than 3,000 lines of code. These systems were chosen to ensure that the bug removal tasks would be neither too difficult, nor too easy for the participants. In addition, the games were chosen for their visual appeal and their potential to engage the participants in removing the bugs.

FB is a Tetris-like game that uses a launcher to shoot bubbles of different colors. A group of three or more of the same colored bubbles can be eliminated by shooting them with another bubble of that color. The aim of the game is to eliminate all bubbles. The bug was designed to represent a visual defect. In both buggy versions (cloned and non-cloned), there are only grey bubbles available. Because all bubbles are the same color, one shot eliminates all of the bubbles. Figure 1 shows the pseudocode for the original code snippet from the *FrozenGame* class. A copy of the snippet shown in Figure 1, existed in class *LaunchBubbleSprite*, which was similar except the third array was missing. For the cloned buggy version of FB, the researchers inserted the bug by changing i+1 to 1 in lines 6 through 8. For the non-cloned version, the copy in the class *LaunchBubbleSprite* was removed. For this version, the *FrozenGame* class had the sole responsibility to load the bubbles and pass them to *LaunchBubbleSprite* upon instantiation. The following bug report describes the problem: *There are eight differently colored bubbles in the game. When a level starts, only gray bubbles appear on top. The launcher at the bottom will also fire only gray bubbles. Obviously not all available bubbles are loaded.*

In PC, the two characters (Pacman and Ghost) are governed by the direction of movement and an invisible grid to check for collisions. The defect was introduced by changing the grid movements for up and left directions. The visual effect of this defect made the characters jump in a flickering motion back and forth. Figure 2 presents the pseudo code for the original movement function. There were copies of this function in the classes *Player* and *Ghost*. The researchers from the original study created the cloned version of PM by changing subtractions to additions in lines 8 and 38 in both the clones. To create the non-cloned version, they abstracted the *switch* structure that was cloned in both classes to the *Actor* class using proper post processing steps. The following bug report describes the problem: *For all game characters, the movement up and left does not work correctly. Instead of moving up or left the characters move in the opposite direction in a flickering motion. Moving down and right works fine for all characters.*

We designed these two systems to emulate real-life bug-fixing scenarios. Bug reports and expected behaviors were provided. The bugs were small and localized, allowing them to be solved within the time constraints. For the cloned bugs, a similar defect was injected in two existing clones in the system. It is to be noted that the code clones in both the software systems were of near-miss type and not exact clones. To produce the non-cloned bugs, these clones were abstracted to a single defective entity. The bugs produced visual behavioral symptoms rather than complete system crashes, making it easier for the participants to reproduce them. In case of the cloned bugs, the two different instances created distinct visual symptoms. Total removal of visual symptoms was possible only after fixing both the instances of the bug.

2) Eclipse Plugin: The original experimenters extended Eclipse with a plug-in that displayed step-by-step instructions (e.g., the task descriptions) to guide the participants through the experiment. The plug-in also logged participant data including the time required to complete tasks and the usage of features like *search*. Finally, the plug-in administered the study surveys.

The environment for performing the study tasks was distributed to participants via USB drives. Each key had preconfigured Eclipse environments for Linux, Mac OS X, and Windows. The subjects unpacked the appropriate archive, ran Eclipse, and loaded the provided workspace. The participants were instructed to save any code changes in the workspace to allow the researchers to later analyze them for correctness. One of the authors supervised the sessions and was available

```
bubbles       = Image[8]
bubblesBlind  = Image[8]
frozenBubbles = Image[8]

for i in 0 .. 8 do
  bubbles[i]  = load("b-"  + toString(i+1) + ".gif")
  bBubbles[i] = load("bb-" + toString(i+1) + ".gif")
  fBubbles[i] = load("fb-" + toString(i+1) + ".gif")
end for
```

Fig. 1. Pseudo Code for FB Code Snippet

to help setup Eclipse. Each participant received the appropriate USB drive for his or her assigned group (see Section II-E).

E. Experimental Operation

We used the two variants of each of the two software systems from in the original study. For clarity we refer to these variants with the same terms as used in the original study, namely: FB_{nc} and FB_c (the non-cloned and cloned versions of FB) and PM_{nc} and PM_c (non-cloned and cloned versions of PC).

Similar to the original study, we divided the participants randomly into Group A and Group B. The objective of partitioning participants into two groups was to counterbalance the study design to remove any potential ordering or interaction effects related to the artifacts. For each of the four tasks (two for the *Replication* and two for the *Extension*), the participants received a version of the system that included one or more bugs. During the *Replication* portion, the participants located and repaired the bugs without any additional documentation aids. However, during the *Extension*, we provided clone reports to the participants to aid in bug location and repair. Table I illustrates which system the participants in each group used for each of the four tasks.

At the end of each session, the Eclipse plug-in created a zip file and saved it back to the USB drive. The zip file contained the four workspaces from the four tasks. Each workspace included: the modified source code, log files and responses to short surveys. We modified the analysis scripts from the original study to work properly on both the *Replication* and the *Extension*.

F. Extended Design

The *Extension* added two bug repair tasks, this time with the benefit of a clone report. This section describes the clone report and requisite training required to use the report.

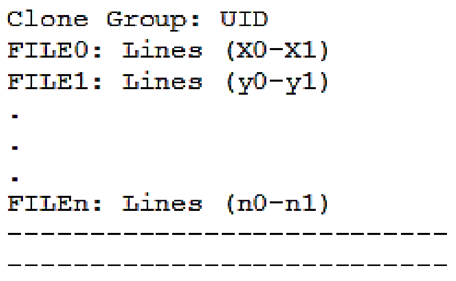
1) Clone Reports: We provided the participants with text-based clone reports similar to those used in our earlier study [4]. The clone reports listed clone groups within the systems as illustrated in the template in Figure 3. Figures 4 and 5 show the specific clone groups that contained the bugs in the FB_c and PM_C systems respectively.

```
switch currentDirection
  case up
    if canMoveTo(gridX, gridY - 1) then
      deltaX = 0
      deltaY = deltaY - speed
      if |deltaY| >= CELL_SIZE then
        deltaY = 0
        moveTo(gridX, gridY - 1)
      end
    end
    break;
  case right
    if canMoveTo(gridX + 1, gridY) then
      deltaX = deltaX + speed
      deltaY = 0
      if |deltaX| >= CELL_SIZE then
        deltaX = 0
        moveTo(gridX + 1, gridY)
      end
    end
    break
  case down
    if canMoveTo(gridX, gridY + 1) then
      deltaX = 0
      deltaY = deltaY + speed
      if |deltaY| >= CELL_SIZE then
        deltaY = 0
        moveTo(gridX, gridY + 1)
      end
    end
    break
  case left
    if canMoveTo(gridX - 1, gridY) then
      deltaX = deltaX - speed
      deltaY = 0
      if |deltaX| >= CELL_SIZE then
        deltaX = 0
        moveTo(gridX - 1, gridY)
      end
    end
    break
end
```

Fig. 2. Pseudo Code for PM Code Snippet

TABLE I
ASSIGNMENT OF GROUPS TO TASK

Sessions	Task #	Group A	Group B
Replication	1	Frozen Bubble w/non-cloned bug (FBnc).	Frozen Bubble w/cloned bug (FBc)
	2	Pacman w/cloned bug (PMc)	Pacman bug w/non-cloned bug (PMnc)
Extension	1	Pacman w/non-cloned bug (PMnc)	Pacman w/cloned bug (PMc)
	2	Frozen Bubble w/cloned bug (FBc)	Frozen Bubble w/non-cloned bug (FBnc)



```
Clone Group: UID
FILE0: Lines (X0-X1)
FILE1: Lines (y0-y1)
.
.
.
FILEn: Lines (n0-n1)
-------------------------
-------------------------
```

Fig. 3. Clone Report Template

2) Training: To avoid biasing the *Replication* portion of the study, we did not inform the participants the subject of the study was code clones. Prior to the *Extension*, we provided a brief background on code clones and how to read a clone report. To prevent biasing the results of the *Extension* portion, we took care only to describe how to read a clone report, now how to use the report for bug removal.

III. RESULTS

This section presents brief results from the original study and detailed results from the *Replication* and *Extension*.

A. Original Study Results

The data obtained in the original study did not provide statistical evidence to reject H_0^{time}or H_0^{corr}. Nevertheless, the results indicated promising trends. Regarding H_0^{time}(RQ1), Table V shows that, although difference was not significant, the participants took more time to fix the cloned bug than they did to fix the non-cloned bug, in most cases. The only exception was that the expert group on average solved PM_c more quickly than they solved PM_{nc}. However, this result may have been caused by the small group size and a uneven distribution of Java skills between the two groups. Overall, it took participants almost twice as long to fix the cloned bug as it took them to fix the non-cloned bug. This difference was primarily caused by subjects who provided incomplete solutions. If only correct solutions are considered and an outlier is removed, the measured difference was much smaller (15.0% for FB and 10.2% for PM). In absolute numbers the difference did not exceed two minutes. The authors concluded that such a difference may not have a considerable effect in practice.

Regarding H_0^{corr}(RQ2), most participants succeeded in fixing at least one bug occurrence. However, for the cloned bug, many participants submitted incomplete solutions. Table VI summarizes the results. Less than half of the students succeeded in fixing both instances of the cloned bug for FB. For PM, one third of the solutions for the cloned bug were incomplete. Surprisingly, some of the experts also failed to fix the cloned bug. For PM_c, one third did not fix both bug occurrences and one expert missed the cloned bug in FB_c. In general, the experts performed better than the students, but the experts participated in the study within a cloning seminar and may have expected clones. The fact that these expert participants provided several incomplete solutions, although they should have been aware of clones, illustrates the risk potential of incomplete fixes of cloned bugs. The authors of the original study concluded that although the results were not statistically significant they indicate that there may be a high risk of such incomplete solutions. These findings encouraged us to replicate the study to further investigate the effect.

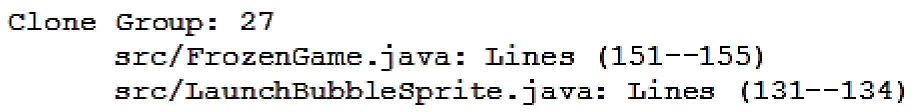
B. Replication

This section describes the results of the *Replication* portion of the replication in three sections: Descriptive Analysis, Timing & Effort, and Correctness.

1) Descriptive Analysis: Each participant was assigned a level of correctness (Addressed, Complete, or Incomplete) for each task based on his or her success in solving that task. As noted earlier, only data from participants that were scored in the Addressed or Complete level of correctness remained in the analysis. The Venn diagram in Figure 6 illustrates the distribution of participants. The set "Cloned Bug — Addressed" represents the participants who solved at least one instance of the cloned bug. This set has a proper subset "Cloned Bug — Complete" that represents the participants who fixed both bug instances. Similarly, the set "Non-Cloned Bug — Complete" represents the participants who fixed the non-cloned bug.

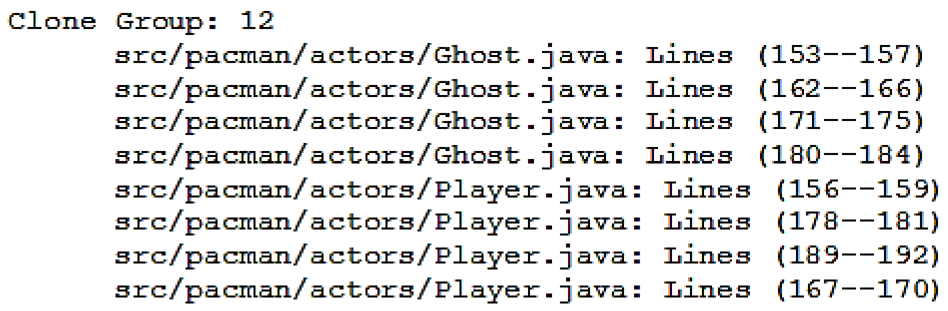
2) Timing & Effort: We used different strategies to measure time and effort. We retrieved these log files to calculate the timing measurements. As there was no direct way to measure effort, we chose to use the Task Load Index (TLX) [9]. TLX is an assessment technique proposed by NASA to assess perceived workload for a particular task. TLX is computed over four different parameters: mental demand (MD), performance (PF), effort (EF) and frustration (FR). We used surveys to gather the following information from participants relative to each TLX parameter:

- mental demand needed to solve the bug,



```
Clone Group: 27
        src/FrozenGame.java: Lines (151--155)
        src/LaunchBubbleSprite.java: Lines (131--134)
```

Fig. 4. Clone Group for FrozenBubble



```
Clone Group: 12
        src/pacman/actors/Ghost.java: Lines (153--157)
        src/pacman/actors/Ghost.java: Lines (162--166)
        src/pacman/actors/Ghost.java: Lines (171--175)
        src/pacman/actors/Ghost.java: Lines (180--184)
        src/pacman/actors/Player.java: Lines (156--159)
        src/pacman/actors/Player.java: Lines (178--181)
        src/pacman/actors/Player.java: Lines (189--192)
        src/pacman/actors/Player.java: Lines (167--170)
```

Fig. 5. Clone Group for Pacman

TABLE II
AVERAGE TIMES AND P-VALUES FOR MANN WHITNEY U TEST

		Frozen Bubble			**Pacman**		
		Cloned		*Non-Cloned*	*Cloned*		*Non-Cloned*
Addressed Participant Times (in seconds)	Mean	1243.7	>	1044.85	491.75	>	460.04
	Median	1135	>	835.5	445.5	>	401
	p-value		0.212			0.432	
Complete Participant Times (in seconds)	Mean	1337.25	>	1044.85	483.92	>	460.04
	Median	1155	>	835.5	445.5	>	401
	p-value		0.238			0.44	

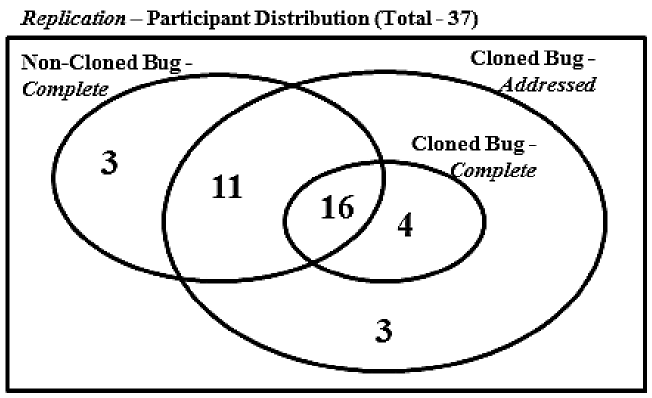

Replication – Participant Distribution (Total - 37)
Non-Cloned Bug - *Complete*
Cloned Bug - *Addressed*
Cloned Bug - *Complete*
3
11
16
4
3

Fig. 6. Distribution of Participants (Replication)

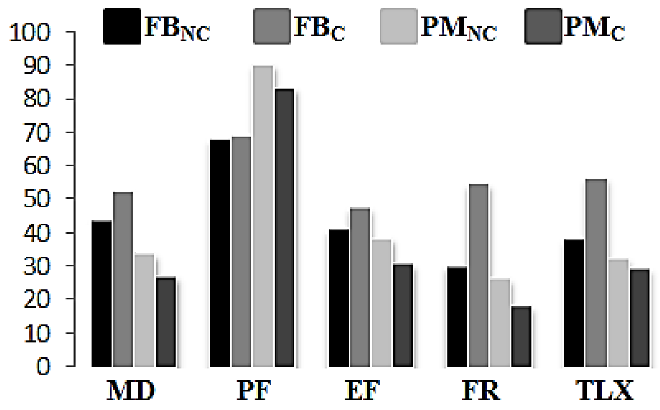


Fig. 7. Task Load Index

- how successful were they performing the task,
- how much effort was needed, and
- how frustrated were they while performing the task.

The results for each artifact, illustrated in Figure 7, show conflicting results. For FB, the TLX scores are higher for the cloned version than for the non-cloned version [TLX FB_{nc} (25.56) < TLX FB_c (39.47)]. This result was expected. Conversely, for PM, the TLX scores are higher for the non-cloned version than for the cloned version [TLX PM_{nc} (26.56) > TLX PM_c (22.02)]. This result was unexpected. Because PM_c had two instances of the bug, we expected the TLX to be higher than for PM_{nc}. We hypothesize that this result may be due to the use of a more complex abstraction to produce the non-cloned version. Further studies are required to check whether there is a definitive reason for this behavior or whether the result is merely an outlier.

To calculate the timing measurements we evaluated the recorded times in the respective workspaces. The Box-and-Whisker plot in Figure 8 illustrates the distribution of times for the *Replication* . Table II provides the mean and median times for the participants. The *Addressed Participant Times*

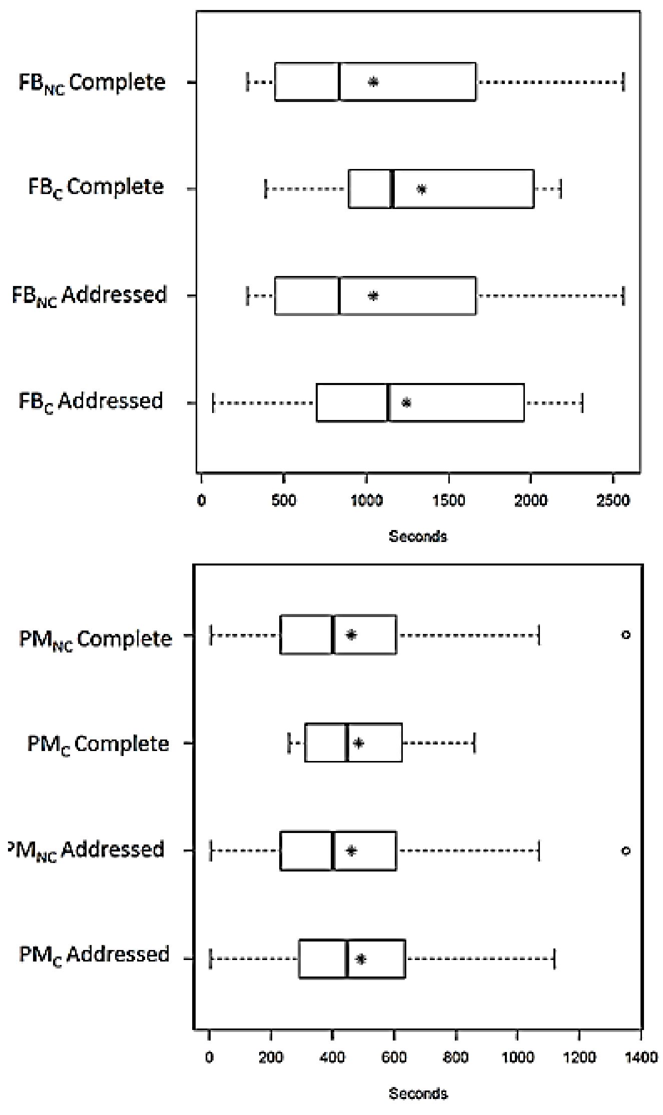


Fig. 8. Times for All Participants in Seconds)

are the times for the participants who fixed at least one bug. The *Complete Participant Times* are the times for participants who found and fixed all the instances of the bug.

Table II shows that the mean and median times were greater for the cloned versions in all cases. The median time to completely fix the cloned bug was 38% greater than the mean time to completely fix the non-cloned in the FB program and 11% longer in the PC program. These differences translate into about 5 minutes for the FB program and just under 1 minute for the PC program. We applied the Mann-Whitney U test to assess whether the independent samples (times for cloned and non-cloned version) were significantly different than each other. Similar to the results of the original study, the differences were not significant (p-value rows in Table II). Hence, though we cannot reject the null hypotheses H_0^{time}, the results do support the same trend that was evident in the original study that the cloned bug took more time to fix than the non-cloned bug.

TABLE III
CORRECTNESS EVALUATION

	Fisher (1-tailed)	Bernard (1-tailed)
FB	< 0.01	< 0.01
PM	< 0.01	0.01

TABLE IV
ADDRESSED VS. COMPLETE FOR CLONED BUG

	Total	Addressed	Complete	%
Replication	37	34	20	59%
Extension	35	35	29	83%

3) Correctness: To judge the correctness of the each participant's solution, we manually checked his or her workspace for bug fixes. First, we executed the programs to check if all the visual symptoms of the bug were removed. Second, we used a diff to inspect the changes to the code. For the *Replication* , 20/37 achieved a Complete solution for the cloned bug, with 30/37 achieving a Complete solution for the non-cloned bug. Fourteen additional participants achieved an Addressed solution for the cloned bug (i.e., they were able to fix only one instance of the bug).

To evaluate H_0^{corr}, we used Barnard's exact test and Fisher's test to test whether the participants were able to obtain a Complete solution to the non-cloned bug more often than they were able to obtain a Complete solution to the cloned bug. These statistical tests are suitable for use on small data sets in the form of contingency tables. The results of these tests, shown in Table III, illustrate that the participants were significantly more likely to obtain a Correct solution to the non-cloned bug. Thus, we can reject the null hypothesis H_0^{corr} and accept the alternate hypothesis H_A^{corr}. This result provides more strength to the results of the original study which showed a promising trend, but not a significant result.

C. Extension

The second part of the replication, the *Extension*, studied the effect of providing developers with information about the clones present in the code being maintained. The *Extension*

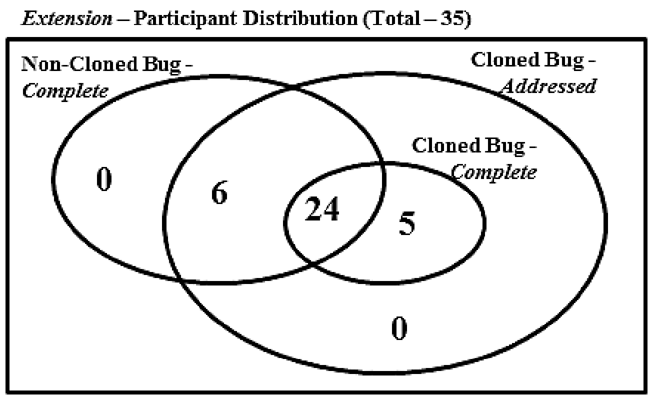

Extension – Participant Distribution (Total – 35)
Non-Cloned Bug - Complete
Cloned Bug - Addressed
Cloned Bug - Complete
0
6
24
5
0

Fig. 9. Distribution of Participants (Extension)

was motivated by RQ3 and tested hypothesis H_0^{cinf} to see whether providing developers clone reports helped them more effectively fix cloned bugs than without clone reports. The Venn Diagram in Figure 9 illustrates the results of the *Extension*.

Similar to the analysis for the *Replication* portion of the experiment, we tested the workspaces collected from each participant for correctness. As shown in Table IV, in this case, all 35 participants Addressed the cloned bug, with 29 providing a Complete solution.

The results from our previous work showed that, if used correctly, clone reports can help developers locate multiple instances of a bug [4]. During the training, we suggested that the participants should employ the correct strategy when using the clone reports, i.e. first locate a bug then use the clone report to determine if there are other instances of that bug. The results shown in Table IV show that the participants in the *Extension* were more successful in locating the cloned bug and obtaining a Complete solution. In the *Extension*, 83% of the participants who fixed at least one instance of the bug (Addressed) were successful in fixing the cloned instance (Complete). This result was an improvement over the *Replication*, where the ratio Addressed:Complete was only 59%. A Chi-Square test on this data showed that the ratio of Addressed:Complete was significantly different between the *Replication* and the *Extension* (X_{21} = 4.267; p = .039). These results allow us to successfully reject the null hypothesis H_0^{cinf} and accept the alternate hypothesis H_A^{cinf}. Therefore, these results indicate that the proper use of clone information can help developers maintain cloned code.

IV. Comparison and Discussion of Results

To gain the most insight from the replication, this section compares the results from the original study to the results from the *Replication* part of the replication study with regards to the time and correctness variables.

A. Time

Table V summarizes the results for the time variable across both studies. In general, it took developers more time to fix the cloned bugs than it did to fix the non-cloned bugs, although these differences were rather small (5 minutes and 1 minute, respectively) and not statistically significant. Across both studies, there was only one case where this result did not hold (experts from the original study working on FB). The authors of the original study did not provide a concrete explanation for why experts behaved differently on the FB problem. They argued that the unexpected behavior may have been an artifact of the small sample size or an uneven distribution of expertise. Our initial hypothesis for this result was that the process used to refactor the clone may have been more complex in the FB program. But, after analyzing the code, the refactoring approach was similar in both systems and the clones seemed to be of similar complexity across both systems.

While cloned bugs tended to take longer to fix, the difference was only significant in one case. Therefore, we are not yet able to draw a definitive conclusion. While some previous research has suggested that clones can be problematic for maintenance, this view is not universal. The results of these two studies seem to confirm that the impact of clones on bug repair still needs further study.

B. Correctness

Table VI summarizes the correctness evaluation for the complete solutions along with the p-values. The results show that for the students in the original study and for the participants in the *Replication*, correctness was significantly worse for the FB program when clones were present. Because the participants in the *Replication* are most similar to the students in the original study, this result is consistent. The results relative to the PC program were inconsistent with a significant result in the *Replication* and a non-significant result in the original study. At this point, we have no concrete explanation for this difference.

Overall, the expert population seemed to behave differently than the students. This result is not surprising because the experts were drawn from the participants in a workshop on code clone research. Due to this characteristic, it is possible that these participants were expecting the system to contain clones. Such an expectation might introduce a participation bias where the traits of the participants can affect the outcome of the study. The students in the original study and the participants in the *Replication* were not aware that the study was focused on code clones, therefore they were likely not biased to look for clones.

In general, the results of the replication are consistent with the results of the original study, with regards to the novice participants. It appears that cloned bugs are more difficult to completely fix correctly.

V. Threats To Validity

This section discusses the threats to the validity for the two parts of the extended replication.

A. Construct Validity

In a real maintenance environment, the process of fixing a bug is iterative. However, in these studies, the Eclipse plug-in was designed so that the participants had to complete one bug fixing task prior to moving on to the next one. They could not go back and revisit their earlier solutions. The dependent variable *correctness* might be affected by this design. It is not clear how the results would be affect if participants were allowed to return to a previous solution. In addition, the dependent variable *time* captured the total time for completion of the task. It was not possible to investigate the time spent on various activities, such as familiarization, playing the games, code comprehension, refactoring. Dividing the time among various activities and only counting those tasks that clearly related to the focus of the study could have produced different results.

TABLE V

Addressed vs. Complete Average Times in Seconds

		FBc		FBnc	p-value	PMc		PMnc	p-value
Students (Original Study)	Addressed	1,615	>	812	0.008	720	>	490	0.175
	Complete	934	>	812	0.22	842	>	490	0.075
Experts (Original Study)	Addressed	923	<	1,276	0.91	838	>	508	0.12
	Complete	516	<	1,276	0.985	825	>	508	0.165
Replication	Addressed	1,244	>	1,045	0.106	492	>	460	0.216
	Complete	1,337	>	1,045	0.119	484	>	460	0.22

TABLE VI

Addressed vs. Complete for Cloned Bug

	Total	Complete	%	Fisher (1-tailed)		Barnard (1-tailed)	
				FB	PM	FB	PM
Students (Original Study)	21	12	57%	0.0085	0.0902	0.0061	0.0617
Experts (Original Study)	12	9	75%	0.5000	0.5000	0.3024	0.3024
Replication	37	20	59%	<0.001	<0.001	<0.001	0.002

B. Internal Validity

We randomly divided the participants into groups without considering their experience or expertise in debugging tasks. There is a chance that the groups were not balanced based on those parameters. Because the participants in each group executed the experimental tasks on different artifacts, it is possible the results would have been different if we had ensured balanced groups. But, because the results of the *Replication* were consistent across artifacts, we do not think this threat is serious. Also, since we excluded the results of participants who were unsuccessful in solving the bugs, the group size was not balanced.

For the *Extension* part, there was the potential for learning bias because we used the same software systems as in the *Replication*. Note that even though the participants used the same systems in the *Extension* as in the *Replication* they were provided with a different variant for each part. The bugs, as explained in Section II, and the refactoring solutions for the two variants of the systems were different and reuse of previous solution was unlikely. We could have reduced this bias by using different participants for the *Extension*, but due to the high cost of human-based studies, this approach was not feasible. We could also have used different systems in the *Extension*, but we would have had a threat that the systems were not similar in complexity. Therefore, we chose to reuse the systems that were developed for the original study. Providing the clone report for the extension part also could have affected the results due to the expectancy of participants to find clones in the system.

We can assume that any learning bias would have affected all participants similarly and therefore should not have affected the results. To reduce this threat to validity we only compared the correctness of the tasks between the *Replication* and the *Extension*. We did not compare the time required because time was inclusive of the familiarization time. Because participants were already familiar with the environment and the system, the times for the *Extension* would have been misleading.

The students participated in the study as a part of classroom assignment. To eliminate any performance biases, the participants were informed that they would be graded on their level of participation in completing all tasks, not in the level of accuracy for the tasks.

C. External Validity

As with any study done using students in a classroom setting, the external validity threats are the most serious. In this case, the software systems we used were smaller than most industrial systems. The participants were relatively novice students compared with industrial practitioners. Finally, the time limit of 75 minutes for the tasks is likely not realistic for an industrial setting. The threats all suggest that the results of these studies must be taken with caution if they are to be applied to any but the most novice of developers. Further studies with experts will help to reduce this threat.

VI. Summary

This extended replication had two parts: (1) a close replication of a previous study and (2) an extension to the previous study. The goal was to provide more insight into the questions from the original study by confirming (or not confirming) its results and by providing additional results that could give insight into the effects of clones on maintenance. The results of the original study showed some trends, but most results were not significant. Specifically, the original study was not able to validate:

1) whether it takes more time to fix a cloned bug than a non-cloned bug, or
2) whether developers are more likely to correctly remove a non-cloned bug than a cloned bug.

While the results of the replication study were also unable to support (1), a similar trend was observed. Conversely, the results of the replication did show that it was significantly more difficult to completely maintain cloned bugs.

In the *Extension* part of the study, we verified that providing developers with clone information and training them on how to use it, helps in the maintenance of cloned bugs. We found that the participants performed significantly better in completely fixing the clone bugs when they had clone information than when they did not. Overall, the evidence from this extended replication indicates that it is more difficult to maintain cloned code than non-cloned code. However, given appropriate clone information and the proper training this difficulty can be reduced.

As a future direction for this research we plan to replicate the study using different sets of participants for the *Replication* and the *Extension* parts, to remove the threat of learning bias. We also plan to investigate the time factor in more detail to determine why the results were not significant. It is possible that measuring only a subset of the tasks involved in the bug fixing process may provide better insight. Another important type of replication is to perform the same experiment with professional developers. Professional developers should be more proficient in debugging tasks. Hence, it will be interesting to compare the times to fix cloned and non-cloned bugs.

Finally, we would like to better understand the relationship between TLX and the difficulty of fixing cloned bugs. In this study we observed anomalous behavior in that the TLX for PC was lower for the cloned bug than the non-cloned bug. We hypothesized that the reason for this result could be the complexity of the abstraction. All else being equal, a cloned bug with multiple instances should have a higher TLX than a non-cloned bug. Although we cannot draw a conclusion from our results, this question might be interesting for future research to gain insight into the ongoing discussion of whether the presence of clones is detrimental to maintenance.

Acknowledgment

We thank the study participants for their time and effort. This material is based upon work supported by the National Science Foundation under Grant No. 0915559.

References

[1] C. Andersson, "A replicated empirical study of a selection method for software reliability growth models," *Empirical Softw. Engg.*, vol. 12, no. 2, pp. 161–182, Apr. 2007.

[2] L. Aversano, L. Cerulo, and M. Di Penta, "How clones are maintained: An empirical study," in *11th European Conference on Software Maintenance and Reengineering*, 2007, pp. 81–90.

[3] N. Bettenburg, W. Shang, W. Ibrahim, B. Adams, Y. Zou, and A. E. Hassan, "An empirical study on inconsistent changes to code clones at release level," in *Proceedings of the 2009 16th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 85–94.

[4] D. Chatterji, J. Carver, B. Massengil, J. Oslin, and N. Kraft, "Measuring the efficacy of code clone information in a bug localization task: An empirical study," in *International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 20–29.

[5] J. R. Cordy, T. R. Dean, and N. Synytskyy, "Practical language-independent detection of near-miss clones," in *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 2004, pp. 1–12.

[6] R. Geiger, B. Fluri, H. C. Gall, and M. Pinzger, "Relation of code clones and change couplings," in *Proceedings of the 9th international conference on Fundamental Approaches to Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 411–425.

[7] J. Harder and R. Tiarks, "A controlled experiment on software clones," in *20th IEEE International Conference on Program Comprehension*, 2012, pp. 219–228.

[8] J. Harder and N. Göde, "Cloned code: stable code," *Journal of Software: Evolution and Process*, pp. n/a–n/a, 2012. [Online]. Available: http://dx.doi.org/10.1002/smr.1551

[9] S. G. Hart and L. E. Staveland, "Development of nasa-tlx (task load index): results of empirical and theoretical research," in *Hancock, P. and Meshkati, N. eds. Human Mental Workload, North-Holland Elsevier Science*, 1988.

[10] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *Proceedings of the 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 485–495.

[11] N. Juristo and S. Vegas, "Using differences among replications of software engineering experiments to gain knowledge," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 356–366.

[12] C. J. Kapser and M. W. Godfrey, ""cloning considered harmful" considered harmful: patterns of cloning in software," *Empirical Softw. Engg.*, vol. 13, no. 6, pp. 645–692, Dec. 2008.

[13] M. Kim, L. Bergman, T. Lau, and D. Notkin, "An ethnographic study of copy and paste programming practices in oopl," in *Proc. 2004 Int. Symposium on Empirical Software Engg.*, 2004, pp. 83–92.

[14] B. Kitchenham, "The role of replications in empirical software engineering–a word of warning," *Empirical Softw. Engg.*, vol. 13, no. 2, pp. 219–221, Apr. 2008.

[15] J. Krinke, "A study of consistent and inconsistent changes to code clones," in *Proceedings of the 14th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 170–178.

[16] ——, "Is cloned code older than non-cloned code?" in *Proceedings of the 5th International Workshop on Software Clones*. New York, NY, USA: ACM, 2011, pp. 28–33.

[17] J. Lung, J. Aranda, S. M. Easterbrook, and G. V. Wilson, "On the difficulty of replicating human subjects studies in software engineering," in *Proceedings of the 30th international conference on Software engineering*. New York, NY, USA: ACM, 2008, pp. 191–200.

[18] M. Mondal, C. K. Roy, M. S. Rahman, R. K. Saha, J. Krinke, and K. A. Schneider, "Comparative stability of cloned and non-cloned code: an empirical study," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2012, pp. 1227–1234.

[19] A. Monden, D. Nakae, T. Kamiya, S. Sato, and K.-i. Matsumoto, "Software quality analysis by code clones in industrial legacy software," in *Proceedings of the 8th IEEE Symposium on Software Metrics*, 2002, pp. 87–94.

[20] F. Rahman, C. Bird, and P. Devanbu, "Clones: What is that smell?" in *7th IEEE Working Conference on Mining Software Repositories*, 2010, pp. 72–81.

[21] F. Shull, V. Basili, J. Carver, J. C. Maldonado, G. H. Travassos, M. Mendonça, and S. Fabbri, "Replicating software engineering experiments: Addressing the tacit knowledge problem," in *Proceedings of the 2002 International Symposium on Empirical Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 7–.

[22] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, "The role of replications in empirical software engineering," *Empirical Softw. Engg.*, vol. 13, no. 2, pp. 211–218, Apr. 2008.

[23] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta, "An empirical study on the maintenance of source code clones," *Empirical Softw. Engg.*, vol. 15, no. 1, pp. 1–34, Feb. 2010.