# Managing Code Clones Using Dynamic Change Tracking and Resolution*

Michiel de Wit, Andy Zaidman, Arie van Deursen

Delft University of Technology
The Netherlands
mail@michieldewit.nl, {a.e.zaidman, arie.vandeursen}@tudelft.nl

## Abstract

*Code cloning is widely recognized as a threat to the maintainability of source code. As such, many clone detection and removal strategies have been proposed. However, some clones can often not be removed easily so other strategies, based on clone management need to be developed. In this paper we describe a clone management strategy based on dynamically inferring clone relations by monitoring clipboard activity. We introduce* CLONEBOARD*, our Eclipse plug-in implementation that is able to track live changes to clones and offers several resolution strategies for inconsistently modified clones. We perform a user study with seven subjects to assess the adequacy, usability and effectiveness of* CLONEBOARD*, the results of which show that developers actually see the added value of such a tool but have strict requirements with respect to its usability.*

## 1 Introduction

Copying source code is a common practice among software developers [18] and is often performed [3] for all sorts of reasons [17]. Duplicated pieces of source code are typically referred to as *'clones'*, which Basit and Jarzabek define as: "code fragments of considerable length and significant similarity" [5]. This definition implies that non-identical, but sufficiently similar pieces of code should also be considered clones. To better capture the differences between identical and very similar clones, various researchers have proposed clone classification schemes, e.g., [7, 4].

Although no clone is the same, it is generally true that clones have a negative impact on the maintainability of the software [17, 21]. A recent experiment by Lozano and Wermelinger [24] shows that cloning often increases the maintenance effort. The following list gives an impression of the sorts of problems that cloning may lead to:

- Being a form of redundancy [15], there is a tendency towards *code growth*, causing longer compilation times.
- By copying and pasting existing code, *bugs* can easily be propagated [18].
- Code may become so heavily duplicated, that developers might start to consider it an *idiom* [17] and will no longer doubt its correctness.

On the other hand, Kapser and Godfrey [17] show that cloning has its benefits as well, e.g., often developers duplicate existing code as a starting point to develop new code more quickly. This technique is dubbed 'forking'. Krinke meanwhile shows that cloned code tends to be more stable than other code as clones are less likely to be edited once they have been created [22].

Studies have shown that software corpora tend to contain large amounts of code clones, ranging from 7% to 23% [20]. As such, the research community has invested a lot of effort into detecting clones [6, 16] and removing clones through refactoring [13, 8]. Kim *et al.* on the other hand suggest that about 50% of clones found in source bases cannot be refactored [19], which opens the door for *code clone management*, i.e., supporting developers in containing code clones and helping them update and maintain cloned fragments.

In this context, Mann [25] suggest to gather clone information while code is written rather than in retrospective as is currently common [21]. Mann proposes to replace the typical copy and paste operations supported by most development environments with well-defined cloning operations so that the developer's duplication intentions can be better modeled. These new operations add meaning to copied fragments, stating whether the copied fragments are meant to remain identical, are subject to minor changes or are copied for other reasons.

This research operationalizes Mann's concept, implements these concepts in an Eclipse plug-in called CLONEBOARD and addresses the following research questions:
**RQ1** Are developers *willing* to alter existing copy and

*This work is described in more detail in the MSc thesis of Michiel de Wit [11].

paste habits to help contain code clones?

**RQ2** In what ways can the relations established by using Mann's operations be used to *enforce* consistent editing of clones?

**RQ3** Will CLONEBOARD help reduce problems related to cloning?

The structure of this paper is as follows: Section 2 introduces the conceptual design of the new copy and paste operators. Section 3 explains our experimental setup, while Section 4 presents the results, which are in turn discussed in Section 5. We go over the threats to validity in Section 6. Section 7 presents related work and Section 8 concludes.

## 2 Conceptual design of Mann's copy & paste operations

Given the extensive use of copy-paste operations and the risks associated with introducing inconsistencies in these cloned fragments, we know that there is a need to help prevent these inconsistencies from being introduced. This understanding is further reinforced by observations from a survey by LaToza *et al.*: 59% of developers feel that finding all instances of duplicated code is a serious problem [23].

In this context, Mann proposes to rethink current editor programs and to implement techniques that help keep redundant code consistent [25]. As it is generally quite hard to differentiate between code duplication that is due to semantic redundancy and duplication that is inherent to the language's grammar, it is necessary to actively include the developer in the process of guarding code consistency. Mann's solution is to replace the *cut*, *copy*, and *paste* operations with a new set of operations that correspond directly to the intended semantics behind their use. With these operations, the user can specify semantic relationships among copied objects, and the editor program can use that information to help in the long-term support of those relationships, thus avoiding inconsistencies [25].

### 2.1 Copy and Paste Scenarios

Mann describes five typical *scenarios* involving the clipboard, he furthermore associates a risk for introducing inconsistencies to each of the five scenarios (see Table 1) [25]. Additionally, these five scenarios help to get a better impression of the reasons developers have for using the clipboard during their programming work.

### 2.2 Mann's Replacement Operators

Mann proposes to counter the risks associated with the aforementioned five copy and paste scenarios (see Table 1), by introducing four operations that are to replace the standard cut, copy and paste operations [25]:

**Move** A code fragment can be moved, similar to the cut and paste command sequence, but removing the code fragment from the clipboard.

**Copy-identical** A code fragment is duplicated and kept synchronous with the original fragment automatically.

**Copy-and-change** Duplication, but with the constraint that the user must change the fragment after duplication.

**Copy-once** This operator copies a fragment, without adding further constraints. As such, this operator emulates the original copy and paste commands.

Mann suggests that the relations established by these operations should be maintained by the development environment automatically. In practice, this means that the copy-identical and the copy-and-change operations need extra attention, whereas the other two operations basically emulate conventional clipboard use.

### 2.3 CloneBoard

We implemented the copy and paste replacement operators in an Eclipse plug-in called CLONEBOARD which is available for download[1]. For our implementation, we interpreted the Mann operations rather flexibly, as we did not actually replace the copy and paste operations, but rather queried the user every time we needed to infer one of Mann's newly proposed operations. This has the benefit that the developers do not need to get used to new shortcuts or menu options.

In practice, this means that when the developer uses the copy and paste commands to introduce a new clone, CLONEBOARD registers the clone relation. When the developer tries to one of the clones in a clone relation, the *clone change resolution* window pops up (see Figure 1), querying the user as to which change resolution he wants to perform on the particular clone instance and/or the whole clone relation. The resolution strategies that the user can choose from are:

- Parameterize clone, i.e., introduce a parameter-token in the cloning relation.
- Unmark clone's tail, i.e., remove tokens from the cloning relation at the end of the fragment.
- Unmark clone's head, i.e., remove tokens from the cloning relation at the start of the fragment.
- Postpone resolution
- Unmark clone, i.e., remove the clone instance from the clone relation.
- Apply changes to all clones
- Ignore changes

Through a set of simple heuristics the resolution strategies are ordered such that the strategy deemed most suitable for the current clone is listed first. Some example heuristics are:

[1] CLONEBOARD homepage at: `http://swerl.tudelft.nl/bin/view/Main/CloneBoard`

| Scenario | Description | Risk of inconsistencies |
|---|---|---|
| Cut to delete | Developers may use the clipboard functionality to delete a specific code fragment, with the assurance that the removed code can be easily restored. | Low |
| Cut/paste to move | The intended use of the cut and paste command sequence is to move code. | Low |
| Copy/paste to duplicate | Code is placed on the clipboard to be replicated at one or more locations later on. | Medium to high |
| Copy/paste to create template | When larger code fragments are duplicated, they are probably not meant to be copied literally, but rather serve as a template for similar code that needs to be written [17]. | High |
| Copy/paste without logical connection | Other reasons may exist for duplicating code, e.g., when programming languages contain a lot of structure elements, chances are that a developer will copy/paste these. | Medium |

**Table 1. Five common copy and paste scenarios and their intrinsic risks.**

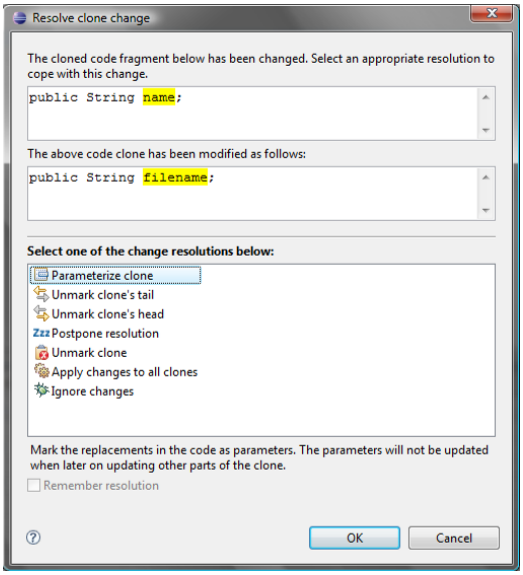



**Figure 1. Part of the CLONEBOARD user interface.**

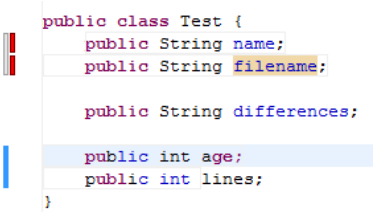



**Figure 2. The CloneBar.**

(1) in order to parameterize clones, the number of changed tokens should be low, (2) for unmarking the clone's tail, the last tokens of the fragment should be changed, (3) postponing is done when changes are being made to comments.

Once a clone relation has been established, CLONEBOARD indicates the presence of clones through the CloneBar (Figure 2 – e.g., a red marker in the left column indicates an inconsistent clone) and the CloneView, a hierarchical overview of clones in the current project.

## 3 Experimental setup

In our experiment we want to verify whether CLONEBOARD is able to change the software development habits of its users. Specifically, we are interested in the opinion of developers regarding the *adequacy*, *usability* and *effectiveness* of CLONEBOARD. For this, we let seven experimental subject work with CLONEBOARD during a number of programming assignments. We employ a *one-group pretest-posttest pre-experimental design* [9]. This type of experiment is called pre-experimental to indicate that it does not meet the scientific standards of experimental design [2], yet it allows to report on facts of real user-behavior, even those observed in under-controlled, limited-sample experiences.

### 3.1 One-group pretest-posttest

In a one-group pretest-posttest pre-experimental design, only one group is tested. Instead of having a control group like in a controlled experiment, the experimental group is subjected to an extra test before the experiment is conducted. This test serves as a baseline to which the measurements gathered after the experiment can be compared. During pretesting and posttesting, the subjects are measured in terms of the dependent variables. Usually, the same questionnaire is used both before and after varying the independent variable (i.e., introducing the tool). By using the same questions, the pretest and posttest results can be compared easier.

For our questionnaires we employ close-end matrix questions in which respondents can rate a number of statements on a 1 to 5 scale, ranging from 'strongly disagree' to 'strongly agree' (the so-called Likert scale).

**Pretest design** For the pretest[2], a total number of five themes were chosen. Each theme relates to a different aspect of the experiment. Most themes are intended to determine possible external variables that might influence the dependent variables, other than the independent variable that is being examined (i.e., the use of CLONEBOARD).

1. Personal background: age, education level and current professional occupation.
2. Development experience: statements related to programming experience and experience with RoboCode, the case for our experiment (see Section 3.2).
3. Attitude towards code quality: statements related to code quality in order to gauge each developer's standpoint towards code quality.

[2]The pre- and posttest design are shown in detail in [11].

**Statement.**
*With a clone management tool, one should be able to see what parts of code have been cloned at any time. Such a tool should give a developer the opportunity to inspect cloning on a per file basis. Furthermore, the tool should alert a developer whenever he is changing a cloned fragment, offering several resolution strategies to cope with the changes. Among such strategies should be the options to update all clone instances.*

**Questions.**
a. Such a tool would significantly help to reduce clone related bugs
b. Interference by such a tool would primarily be inconvenient
c. A clone management tool will save me a lot of time
d. I dont see the added value of such a tool
e. I expect to be making use of this tool quite extensively
f. The tool will not be able to solve real problems

**Figure 3. Question 5 of the pretest.**

4. Attitude towards cloning: a number of statements that gauge each subject's familiarity with the concept of cloning were added.
5. Expectations of a code clone management tool (after introducing such a tool very abstractly – see Figure 3).

**Posttest design** After the subjects have completed their assignments, they have to fill out a second questionnaire serving as a posttest, of which the primary intent is to measure whether the subjects' expectations with regard to a clone management system have been fulfilled and if CLONEBOARD is found to be a useful example of such a tool.

In the posttest, a number of different issues are addressed. First of all, a number of checks are performed to verify that the experiment went well:

I Assignments experience: were the assignments too hard, was there time pressure?

II Development style: we verify whether the subjects followed the coding habits that the subjects indicated in question 3 of the pretest.

We also make inquiries to assess each subject's experiences with the CLONEBOARD user interface. These questions are intended to measure to what extent the subject noticed the presence of CLONEBOARD and actively used it.

III UI experience: some general questions on the subject's interaction with CLONEBOARD.

IV Resolution window experience: statements relating specifically to the clone change resolution window.

To get an impression of the subject's perception of the change resolutions, we pose the following questions:

V Resolution frequency: the subjects are asked to indicate how many times they used each of the seven resolutions: never, once, 2–5 times, 6–10 times, or more than 10 times. Given the relatively short time given for the assignments and existing figures about developer copy and paste usage [18], this scale seems to be fair.

VI Resolution value: the usefulness of each of the resolutions on a Likert scale is asked for. If a resolution is never used, the participants are asked to indicate how useful they think the resolution would be.

The seventh question of the questionnaire addresses the dependent variable. Subjects were given the same seven statements as used in question 5 of the pretest (see Figure 3), only changed to feature the name of the clone management tool. Next, we can compare the answers for this question to those given in the pretest, to see to what extent CLONEBOARD meets the subjects' expectations.

To further measure the participants' perception of the CLONEBOARD user interface and to see to what extent problems in the usability of the tool hindered its use, we added an extra series of Likert-scale statements.

As a means to assert that problems with the case, the execution of the experiment, its documentation or the questionnaires did not have a negative effect on its validity, subjects were asked to rate a final series of statements. Instead of a Likert scale, a scale from 1 to 9 is used to enable participants to give a more fine-grained answer.

We also left enough space for participants to write down some comments or suggestions they might have.

## 3.2 RoboCode

To test its influence on the dependent variables, subjects will have to be exposed to CLONEBOARD, preferably in a way that maximizes generalizability of the final results. We do this by simulating a development task in an environment that includes CLONEBOARD. However, simulating a development task is not easy. For this experiment, we picked an existing software system as a case and give subjects a number of programming assignments that involve modifying existing source code. For this approach to be successful, a number of important conditions have to be met: (1) it should involve a realistic, sufficiently complex case, (2) the case should be interesting to get test subjects involved, (3) participants should be equally familiar with the case, and (4) the case should be relatively easy to learn.

After careful consideration, RoboCode[3] was selected as the case for our experiment. RoboCode is an artificial intelligent (AI) programming puzzle. RoboCode implements a simulation framework in which several artificially intelligent agents are confronted to each other to find the one with the best logic. Each of the agents represents a robot, that is put into an arena with other robots in a struggle for victory.

## 3.3 Programming assignments

We created a basic robot implementation in RoboCode, as an example for the programmers. We then designed five programming assignments, for which we attempted to

[3] See `http://robocode.sourceforge.net/`

include tasks that would provoke code cloning. Specifically, we reused a number of motives for cloning described in literature, e.g., reuse of complicated control structures [18, 17] and the lack of language support for secondary concerns [26].

In the first two assignments, the subjects were asked to implement logging functionality. These assignments both help participants to get to know the software better and are likely to give rise to code cloning. The third assignment requires the developers to implement a series of variations on an already implemented target selection algorithm. The variations were designed not to be too large, making them excellent candidates for code duplication. Finally, the fifth assignment more or less gives participants *carte blanche* to extend and alter the robot code as they see fit. The final list of assignments used in the experiment is detailed in [11].

### 3.4 Pilot

To test run the experiment and locate any problems in its design, a pilot study with one subject was performed prior to the actual experiment. The pilot pointed us towards a number of issues, in particular, we made the programming assignments a little bit easier so that they would fit a two-hour experiment. Furthermore, some bugs in CLONEBOARD were discovered and fixed.

## 4 Results

In this section we discuss the data obtained from the pretest and the posttest. We defer the analysis of the results to Section 5.

### 4.1 Subject Profile

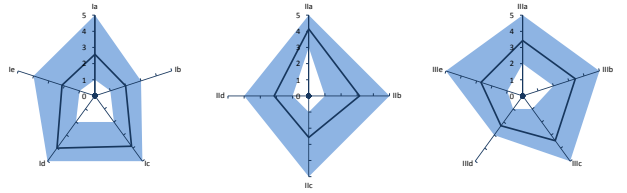
The seven volunteers that participated in our experiment were recruited within the computer science faculty, and as such they all had either an MSc degree, PhD degree or were very close to one. Furthermore, all participants were male and had ages between 22 and 29. Three out of the seven volunteers were working (part-time) as a software developer.

Most subjects consider themselves averagely experienced (average score 3.4) and rather proficient (average score 3.9) Java developers. For most, Eclipse is their 'native' development environment. None of the subjects had prior knowledge of RoboCode.

The subjects' attitude towards code quality is fairly consistent (cf. the radar diagram in Figure 4a; each branch represents a single question; the colored surface indicates the range of the answers given; the bold line shows the average) as writing clean code is valued about equally as writing functional code (question *3a* in Figure 4a). Most respondents agree that bugs are often the result of programmer


(a) Attitude towards code quality (b) Attitude towards cloning

**Figure 4. Subject profile.**




(a) Assignments experience (b) Development style (c) UI experience

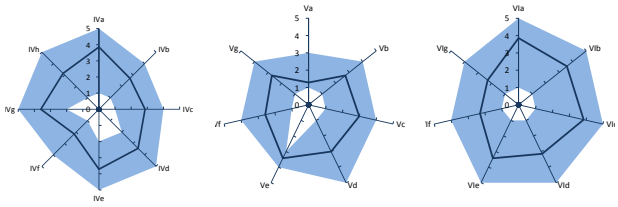
**Figure 5. Subjects' experiences with CLONEBOARD and the assignments.**

sloppiness (*3b*, average score 3.7). All developers endorse a focus on writing good quality code (*3c*, average score 4.0). Regarding commenting behavior, the subjects indicated that they generally keep a fair balance between the amount of code and comments they write, with a slight bias towards code (*3d*, average 2.3). With only one subject rating the statement lower than 3, respondents seemed to believe that better tools can actually prevent bugs (*3e*, average 3.6). Only few use Eclipse's code refactoring facilities often (*3f*, average 2.1).

With the exception of one respondent, all subjects were very familiar with the concept of cloning (*4a* in Figure 4b) and agree that copy and pasting is not the best reuse strategy, neither in general (*4c*), nor when it comes to crosscutting concerns (*4d*). Copy and paste habits seem to differ: some indicate to copy and paste a lot while programming, whereas others are more reluctant copiers (*4b*, average 3.1). Most respondents have come across inconsistent clones, but none did so very often (*4e*, average 3.1). Apart from two subjects, all agreed that cloning can lead to bugs (*4f*, average 3.6).

### 4.2 Working with CloneBoard

The first three questions of the posttest questionnaire (Figure 5) are used to get a basic impression of the subject's experiences with the assignments and CLONEBOARD. These questions help to assert that the assignments were adequate to provoke the desired kind of behavior. The results (cf. Figure 5) show that the participants generally did not find the assignments too hard (*Ia*, average 2.6). None of the subjects experienced time pressure and actually even tended towards the inverse (*Ib*). The respondents reported



(a) Resolution experience (b) Resolution frequency (c) Resolution value

**Figure 6. Subjects' experiences with the clone change resolutions.**

that they found the assignments interesting to do (*Ic* and *Id*). With the exception of one, all subjects were satisfied with the level of a priori information provided (*Ie*, average 2.1).

Concerning development style, the respondents all gave rather different answers. Although all subjects confirmed that their programming work reflected their usual habits (*IIa*), some reported to have focused on functional code more than others (*IIb*) and the reported degree of comments written varies greatly (*IIc*). Most participants indicated not to have copied and pasted more than they would normally do, some reported to have copied slightly less (*IId*).

When asked about the subjects' general experiences with CLONEBOARD, nearly all respondents reported to have encountered CLONEBOARD during their assignments, although not all indicate they did so often (*IIIa*). Interesting to note is that the respondents seem to identify the change resolution window with CLONEBOARD, as all of them replied the same to both statements *IIIa* and *IIIb*, the latter of which asks about encounters with the resolution window.

Quite a few subjects reported to have often quickly dismissed the resolution window by canceling it (*IIIc*, average 3.4, median 4).[4] Both the CloneView (*IIId*) and CloneBar (*IIIe*) were not rated very high: with the exception of one respondent, all indicated to find little use for these two navigational elements.

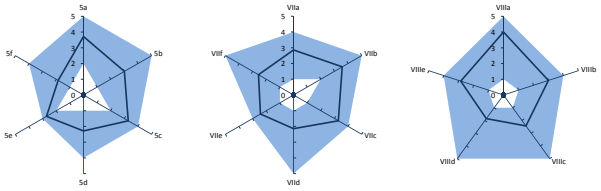
## 4.3 Resolutions

During the experiment, all subjects were confronted with the clone change resolution window, that would pop up after a clone had been changed inconsistently. Figure 6 provides an overview of the subjects' experiences with the clone resolutions. To most subjects it was clear most of the times why the window appeared (*IVa*, average 3.9). The window clearly did not always show at convenient moments (*IVb*, average 2.7). For some, the *before* and *after* views of the changed clone fragment were not sufficiently clear (*IVc*, average 2.9), probably because they were too small at times, as commented by one of the respondents.

[4]One respondent denied cancelling the window, but rather indicated that he had confirmed it blindly, most of the time. His original rating to *IIIc* was changed from 1 to 5, as canceling and blindly confirming can be considered the same in the context of the question.

| Resolution | Usage | Value |
|---|---|---|
| Apply changes to all clones | Almost never | Very useful |
| Ignore changes | Fairly often | Useful |
| Parameterize clone | Quite often | Very useful |
| Postpone resolution | Quite often | Fairly useful |
| Unmark clone | Quite often | Useful |
| Unmark clone's head | Fairly often | Not so very useful |
| Unmark clone's tail | Fairly often | Not so very useful |

**Table 2. The experimental subjects' opinions about the 7 clone change resolutions.**



(a) Expectations (b) Perception (c) UI problems

**Figure 7. Subjects' evaluation of CLONEBOARD as a clone management tool.**

Most of the developers more or less agreed that the window showed sufficient information (*IVd*, average 3.4) and that the order of the resolutions was quite logical (*IVe*, average 3.7). The *Remember resolution* option of the window was not valued well (*IVf*), but it was clear to most participants why the option was not always available (*IVg*, average 3.6).[5] Some of the participants missed change resolutions (*IVg*), most notably more advanced parameterization and refactoring operations.

The respondents were asked how often they used each of the resolutions and how they valued each of them (*V* and *VI*). Table 2 summarizes the results, showing the most useful resolutions according to the test subjects. Strikingly, none of the subjects actually used the *Apply changes to all clones* resolution, yet it is valued highest of all. One participant comments about this, stating that the case did provoke cloning, but gave little reason to update clones.

## 4.4 Tool Evaluation

The most important questions of the experiment are question 5 of the pretest and question VII of the posttest. In these questions, CLONEBOARD is compared to the expectations the respondents had of a hypothetical clone management tool with CLONEBOARD's functionality. These questions, together with posttest question VIII measure the dependent variables of the experiment.

In Figure 7 the radar charts show some differences between the participants' original expectations and their perception of CLONEBOARD. These differences can be observed better in Figure 8, in which the averages of the ratings are shown. Although participants are still not very convinced a clone management tool would save them time (*5c*

[5]Two of the respondents actually did not rate these statements as they had not noticed the option at all.

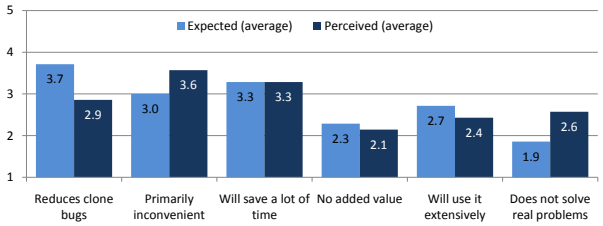



**Figure 8. Expectations for and perceptions of** CLONEBOARD**.**

and *VIIc*), CLONEBOARD apparently offers slightly more added value than the respondents expected (*5d* and *VIId*).

The test subjects seem to be somewhat disappointed with CLONEBOARD, in the sense that they had expected that it would help reduce clone related bugs better (*5a* and *VIIa*). This outcome might be slightly colored by two of the subjects, who changed their opinions rather radically. The higher than average medians of the ratings illustrate this fact. Furthermore, the respondents do not seem so very convinced anymore that CLONEBOARD as a clone management tool might solve any real problems (*5f* and *VIIf*). This insight is shared rather broadly among the test subjects. Only one respondent indicates that CLONEBOARD is more likely to solve problems than he had expected.

Also, there is a rather large difference between the expected and perceived inconvenience of CLONEBOARD (*5b* and *VIIb*). Whereas participants were rather mild about this in the pretest, after using CLONEBOARD, their attitude has changed significantly. This perceived inconvenience probably let them to indicate a lower chance of actually using the tool in practice (*5e* and *VIIe*).

CLONEBOARD's user-interface elements do not seem to be the reason of the subjects' disappointment. In general, participants found the UI easy to use (*VIIIa*, average 4.0) and are fairly neutral when it comes to assessing their ability to get used to CLONEBOARD as part of their IDE (*VIIIb*). The respondents reported some errors (*VIIIc*, average 2.4), but these apparently did not hinder CLONEBOARD's functionality significantly (*VIIId*, average 1.9).

When asked whether CLONEBOARD would need a better user-interface, opinions seem to differ. Some are more negative than others, but on average, subjects remain fairly neutral (*VIIIe*, average 2.9).

# 5 Discussion

The purpose of the conducted experiment was to investigate three variables relating to CLONEBOARD: *adequacy*, *usability* and *effectiveness*. In the quesionnaires of the pretest and posttest, specific questions were asked to measure these variables. We will now analyze the results of these questions to determine to what extent CLONEBOARD is adequate in managing code clones.

## 5.1 Adequacy

To find out whether CLONEBOARD as a tool is sufficiently adequate for developers to use as a clone management solution in their daily work, we posed three pairs of statements in pretest question 5 and posttest question VII:

***5c & VIIc*** CLONEBOARD will help me save a lot of time.

***5d & VIId*** I do not see the added value of CLONEBOARD.

***5e & VIIe*** I expect that I would be making use of CLONEBOARD quite extensively.

***VIIIb*** I will be able to get used to using CLONEBOARD in everyday coding.

The results of the experiment show that the respondents do see added value for a tool like CLONEBOARD, actually even more than they initially expected. Although the difference is small, it does show that CLONEBOARD lives up to the respondents expectations with regard to its potential to add value to the development process. Actually, one respondent was heavily disappointed: he had great expectations, but apparently perceived the opposite.

Before the experiment, the test subjects were not convinced of a clone management tool's ability to save them time, and this opinion did not greatly change. Some were more optimistic than others, but apparently CLONEBOARD either is not able to save developers time or the experiment was too short to accurately assess time savings.

**Reflection.** Considering these results and the fact that respondents were not overly optimistic about their ability to get used to CLONEBOARD in their daily practice, it seems fair to doubt CLONEBOARD's adequacy. Although it clearly does add value, its adequacy as a tool has not been shown. A longitudinal study would be required to draw more precise conclusions.

## 5.2 Usability

While CLONEBOARD was reported to be easy to use, nearly all respondents claimed that they found CLONEBOARD slightly too obtrusive. The answers to statement *VIIIa* clearly illustrate this (Figure 7). Only half of the respondents is convinced that CLONEBOARD would need a better user interface to be useful (statement *VIIIf*).

The clone change resolution window seems to be the main cause for irritation. Some of the respondents reported to have mainly dismissed the resolution window by pressing the *cancel* button or hitting *escape* (*IIIc*). The window did not always pop up at convenient moments (*IVb*). Comments given by some of the respondents learned that improving the timing will probably not help, as it is just the *concept* of a pop-up window that annoys developers: a less obtrusive query mode (e.g., a warning icon or message in a sidebar) would probably lead to less disturbance.

About the resolution window itself, respondents were quite positive; it shows sufficient information to make its purpose clear (*IVa* and *IVd*) and the presentation order of the resolutions was found logical by most respondents (*IVe*).

Other parts of the CLONEBOARD user interface (i.e., the CloneView and CloneBar) were not found to be especially useful (*IIId* and *IIIe*). The size of the case probably did not require the use of these controls to navigate the clone model.

**Reflection.** In short, the experimental subjects found CLONEBOARD to be usable, but too obtrusive.

### 5.3 Effectiveness

To get an impression of CLONEBOARD's effectiveness subjects were asked the following question: will CLONEBOARD solve problems (*5f* and *VIIf*)? In the pretest, the respondents were quite optimistic about this. Two hours later, however, the subjects were less so.

A similar decline of optimism is shown by the answers subjects gave when asked about CLONEBOARD's potential to reduce clone related bugs. Expectations about this were rather high (*5a*), but were less so after the experiment.

**Reflection.** So, is CLONEBOARD effective? The experimental subjects remained rather neutral about CLONEBOARD's effectiveness, suggesting that a more elaborate evaluation, e.g., using a controlled experiment, will be required for a conclusive answer.

### 5.4 Usefulness of the Resolution Mechanism

Although the primary objective of the experiment was to assess CLONEBOARD's value as a clone management tool, the questionnaire results that relate to the resolution strategies used were such that they justify further analysis.

It was interesting to see that the change resolutions that were used least were valued the most (cf. Table 2). Most notably, the *Apply changes to all clones* (cf. Section 2.3) resolution was applied only three times, but rated highest. Quite in contrast with this, another highly valued resolution, *Parameterize clone*, was actually used very often. The other resolution strategies were valued less, indicating that the primary interest of developers in a clone management tool is to be able to keep similar code fragments in sync and patch them whenever a bug requires so.

**Reflection.** This observation adds to the impression that a clone management tool should primarily guard and enforce clone integrity, but should not bother developers with the details of what exactly defines a clone and what parts should be included. The experimental subjects showed to be quite willing to define parameterized clones, but often escaped the hassle of other administrative tasks by dismissing CLONEBOARD's queries or removing clone markers altogether to prevent being disturbed.

## 6 Threats to Validity

In this section we discuss the validity of the observations resulting from the experiment. We consider two types of validity: internal validity, in which we touch upon cause-effect inferences made during the analysis, and external validity, which relates to the results' generalizability.

### 6.1 Internal Validity

Analysis of the experiment's outcomes relies on the assumption that the only factor influencing the dependent variables is CLONEBOARD itself. However, several factors relating to the subjects and the circumstances may have interfered. First of all, some subjects may have felt (emotional) pressure to answer positively. In response, we attempted to make clear to all subjects that they did not have to please anybody, as only sincere answers were of value.

The assignments may have induced certain types of behavior with the subjects, e.g., paying more attention to cloning. Control questions have been added to the posttest to test for these effects. From the subjects' answers we observed that the selected case and assignments were found suitable.

The duration of the experiment may have influenced the validity of the results, as well. A duration of two hours was considered somewhat short by some of the participants, especially as it is hard to appreciate long-term benefits of clone management software in such a short time. To counter this effect, participants were deliberately asked to try and consider how the software would work for them in practice.

### 6.2 External Validity

Generalizability of the experiment's results depends mainly on three major aspects: representativeness of the subjects, suitability of the selected case and the degree of similarity the programming assignments bear to real-world development tasks.

Although all subjects were academics, their backgrounds were sufficiently different to assume they represent average developers. Some of the subjects had significant commercial development experience, whereas others had a more theoretical background.

A clear disadvantage of selecting a case that is easily comprehensible, is that the case is probably also not so very complex. Real-life systems are likely to be more complex than RoboCode. CLONEBOARD might prove to be more useful in a more complex system. However, all participants

were asked to what extent they expected the tool to be of use in their daily practice.

One of the subjects remarked that the programming assignments were more focused on inducing clone creation than on clone modification. Given the rather limited time frame, it is hard to simulate the effect of modifying clones other developers created. This shortcoming may have had a significant influence on the outcomes of the experiment. That is why we see a longitudinal study as the next necessary step in this line of research.

## 7 Related work

Code clone research has traditionally focused on clone detectors [6, 16]. It is only more recently that effective ways to actually manage code cloning have been researched [1]. This section gives a brief overview of some recent advances in the area of clone management.

Toomim *et al.* investigate the concept of *Linked Editing* to simultaneously update clone fragments [27]. CLONEBOARD includes some of Toomim *et al.* 's ideas, e.g., cascading a change to one clone to the rest of the clone set. Toomim *et al.* 's experiment showed that linked editing can save a lot of time when compared to the more traditional approach of functional abstraction to refactor redundant duplicates.

Jablonski and Hou have developed a framework that captures copy and paste activity and uses this information to dynamically track clones [14]. Their software, dubbed *CnP*, automatically links identifiers, so that rename operations can be guaranteed to be performed consistently. In essence, this approach is very similar to CLONEBOARD's. The main difference is in the way changes are handled: whereas *CnP* only assists in rename operations, CLONEBOARD tries to approach changes in a broader sense.

Duala-Ekoko and Robillard bring several clone management techniques together in a tool called *CloneTracker* [12]. This Eclipse plug-in maintains a model of all clones in a source base. The data for this model is gathered using a third-party clone detector. The way in which *CloneTracker* visualizes clones and allows navigation is similar to the techniques employed by CLONEBOARD. When it comes to handling clone changes, however, *CloneTracker* rather resorts to using linked editing, whereas CLONEBOARD introduces the concept of automatic change resolutions.

Two tools that are very similar to CLONEBOARD in terms of technology are *Clonescape* by Chiu and Hirtle [10] and *CPC* by Weckerle [28]. Both tools are Eclipse plug-ins that monitor clipboard activity to infer clone relations, similar to the way CLONEBOARD does. *Clonescape*, however, focuses more on providing clone navigation tools, while *CPC* was implemented to be a framework for others to base clone management technology on. The main difference in functionality between CLONEBOARD and *CPC* being that *CPC* only notifies users of possible clone inconsistencies, not offering resolutions for clone management. The *CPC* tool was tested by some developers and the most important conclusion of that user study was that the tool improved awareness of the cloning situation within a software system.

## 8 Conclusion

In this paper, we have introduced CLONEBOARD, an Eclipse plug-in that dynamically tracks code clones by monitoring the clipboard activity. CLONEBOARD features clone change resolutions, which assist the developers with managing their code clones, e.g., by applying changes made to a particular clone to all clones in the clone set or by parameterizing a clone relation. In order to assess the usefulness of CLONEBOARD we have conducted a pretest-posttest experiment involving seven developers, in which we presented the developers a questionnaire before and after a two-hour programming assignment in an environment in which they were faced with CLONEBOARD.

Our results show that developers actually see the added value of code clone management tools, but have strict requirements with respect to their usability. We will now go over the research questions that we have stated in Section 1:

**RQ1** *Are developers willing to alter existing copy and paste habits to help contain code clones?* Apart from the fact that CLONEBOARD did not always intervene at convenient moments, respondents were optimistic about the concept of clone change resolutions. It was also clear to most that automated support for managing clones can save them time.

**RQ2** *In what ways can the relations established by using Mann's operations be used to enforce consistent editing of clones?* The clone change resolutions implemented in CLONEBOARD allow to forward changes made in one clone to all other instances. The respondents highly appreciated this feature.

**RQ3** *Will* CLONEBOARD *help reduce cloning related problems?* The respondents made it clear that CLONEBOARD would only be of limited use to them, mainly because they are expecting more advanced resolution strategies. In essence, this does not answer our question of whether Mann's operations help in reducing cloning related problems.

**Contributions.** Over the course of this research we have made the following contributions:

- The freely downloadable CLONEBOARD Eclipse plug-in, which features dynamic clone tracking by means of monitoring clipboard activity.
- A series of clone change resolutions, which go beyond removal and refactoring of code clones, and which of-

fer hints at the best way of dealing with a clone through the order in which they are presented.
- A user study with seven developers to assess the adequacy, usability and effectiveness of CLONEBOARD.

**Future work.** Our most important avenue for future work is to extend the current study with a *longitudinal study* in order to properly assess the value and effectiveness of clone management solutions in general and CLONEBOARD in particular. Such a study would involve studying the cloning-related habits of several CLONEBOARD-users over a period of weeks or even months. We also plan to extend our resolution strategies with more advanced options and make the resolution window less obtrusive, e.g., by implementing it as a warning message.

**Acknowledgments.** Our thanks go out to the volunteers for our experiment and to Cathal Boogerd and Bas Cornelissen for proofreading this paper. This research was sponsored by the NWO Jacquard Reconstructor project and by the NIRICT Centre of excellence for Dependable ICT Systems CeDICT.

## References

[1] L. Aversano, L. Cerulo, and M. Di Penta. How clones are maintained: An empirical study. In *Conf. Softw. Maintenance and Reengineering (CSMR)*, pages 81–90. IEEE, 2007.

[2] E. Babbie. *The practice of social research*. Wadsworth Belmont, 11th edition, 2007.

[3] B. Baker. On finding duplication and near-duplication in large software systems. In *Proc. Working Conference on Reverse Engineering (WCRE)*, pages 86–95. IEEE, 1995.

[4] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis. Measuring clone based reengineering opportunities. In *Proc. of the Int'l Symposium on Software Metrics (METRICS)*, page 292. IEEE, 1999.

[5] H. A. Basit and S. Jarzabek. Efficient token based clone detection with flexible tokenization. In *Proc. of the the joint meeting of the European Software Engineering Conference and the symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 513–516. ACM, 2007.

[6] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Int'l Conf. on Software Maintenance (ICSM)*, pages 368–377. IEEE, 1998.

[7] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *Transactions on Software Engineering*, 33(9):577–591, 2007.

[8] M. Bruntink, A. van Deursen, R. van Engelen, and T. Tourwé. On the use of clone detection for identifying crosscutting concern code. *Transactions on Software Engineering*, 31(10):804–818, 2005.

[9] D. Campbell, J. Stanley, and N. Gage. *Experimental and quasi-experimental designs for research*. Rand McNally Chicago, 1963.

[10] A. Chiu and D. Hirtle. Beyond clone detection. Technical report, University of Waterloo, 2007.

[11] M. de Wit. Managing clones using dynamic change tracking and resolution. Master's thesis, Software Engineering Research Group, Delft University of Technology, 2009.

[12] E. Duala-Ekoko and M. Robillard. Clonetracker: Tool support for code clone management. In *Proc. Int'l Conf. on Software Engineering (ICSE)*, pages 843–846. ACM, 2008.

[13] R. Fanta and V. Rajlich. Removing clones from the code. *Journal of Software Maintenance*, 11(4):223–243, 1999.

[14] P. Jablonski and D. Hou. CReN: a tool for tracking copy-and-paste code clones and renaming identifiers consistently in the IDE. In *Proc. of the OOPSLA workshop on Eclipse technology eXchange*, pages 16–20. ACM, 2007.

[15] J. H. Johnson. Identifying redundancy in source code using fingerprints. In *Proceedings of the conference of the Centre for Advanced Studies on Collaborative research (CASCON)*, pages 171–183. IBM Press, 1993.

[16] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *Transactions on Software Engineering*, 28(7):654–670, 2002.

[17] C. Kapser and M. Godfrey. "Cloning Considered Harmful" considered harmful. In *Proc. Working Conference on Reverse Engineering (WCRE)*, pages 19–28. IEEE, 2006.

[18] M. Kim, L. Bergman, T. Lau, and D. Notkin. An ethnographic study of copy and paste programming practices in OOPL. In *Proc. of the Int'l Symposium on Empirical Software Engineering (ISESE)*, pages 83–92. IEEE, 2004.

[19] M. Kim, V. Sazawal, D. Notkin, and G. Murphy. An empirical study of code clone genealogies. *SIGSOFT Softw. Eng. Notes*, 30(5):187–196, 2005.

[20] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *Proc. of the Int'l Symposium on Static Analysis (SAS)*, pages 40–56. Springer-Verlag, 2001.

[21] R. Koschke. Identifying and removing software clones. In *Software Evolution*, chapter 2, pages 15–36. Springer, 2008.

[22] J. Krinke. Is cloned code more stable than non-cloned code? In *Proc. of the Int'l Working Conf. on Source Code Analysis and Manipulation (SCAM)*, pages 57–66. IEEE, Sept. 2008.

[23] T. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *Int'l Conf. Softw. Engineering (ICSE)*, pages 492–501. ACM, 2006.

[24] A. Lozano and M. Wermelinger. Assessing the effect of clones on changeability. In *Proc. Int'l Conference on Software Maintenance (ICSM)*, pages 227–236. IEEE, 2008.

[25] Z. Mann. Three public enemies: cut, copy, and paste. *Computer*, 39(7):31–35, 2006.

[26] P. Tarr, H. Ossher, W. Harrison, and S. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *Int'l Conf. Software Engineering (ICSE)*, pages 107–119. ACM, 1999.

[27] M. Toomim, A. Begel, and S. L. Graham. Managing duplicated code with linked editing. In *Proc. of the Symposium on Visual Languages - Human Centric Computing (VLHCC)*, pages 173–180. IEEE, 2004.

[28] V. Weckerle. Cpc an eclipse framework for automated clone life cycle tracking and update anomaly detection. Master's thesis, Freie Universität Berlin, January 2008.