

A Comprehensive Evaluation of Workspace Awareness in Software Configuration Management Systems

Anita Sarma, André van der Hoek, and David F. Redmiles
University of California, Irvine
Irvine, CA 92697-3440 U.S.A.
{*asarma, andre, redmiles*}@ics.uci.edu

Abstract

Workspace awareness has emerged as a new coordination paradigm in configuration management, enabling early detection of potential conflicts by providing developers with information of relevant, parallel activities. The focus of our work is on detecting and mitigating direct and indirect conflicts by sharing information about ongoing code changes. In this paper, we discuss the results of user experiments designed as a broad and formative evaluation of workspace awareness, specifically focusing on whether users detect conflicts as they arise and act to mitigate potential problems. Our results confirm that workspace awareness promotes active self-coordination among users and can lead to an improved end-product in terms of the number of unresolved conflicts remaining in the code.

1. Introduction

Configuration Management (CM) systems have become one of the most popular and widely adopted coordination tools in the software industry [1]. To facilitate multiple developers working on a common set of artifacts, CM systems depend on a central repository that has a well-defined access and synchronization protocol. In a typical scenario, developers check out artifacts from this repository into private workspaces and, once their changes are complete, put their changes back into the repository.

Private workspaces are essential in allowing developers to work without interference, but have the negative side effect of hiding team members' activities. As a result, developers work in an isolated manner and perform their work without knowing whether or not it is conflicting with changes ongoing in other workspaces. Conflicts are thus detected later, only after developers have finished their changes and attempt to check them in. Furthermore, only *Direct Conflicts*, which arise due to changes to the same artifact, can be detected by CM

systems. *Indirect Conflicts*, which arise when changes in one artifact affect concurrent changes in one or more other artifacts, remain undetected until the system is built, tested, or more problematically so, during actual use at a customer site. The resolution of conflicts that are detected so much later than when they are actually introduced is expensive and time consuming [2, 3].

One way to overcome this problem is to inform developers of ongoing activities that are deemed relevant to the developer's current tasks and the effects of these activities on the local workspace. Developers can then place their work in the context of others' changes and self-coordinate their actions to resolve conflicts right when they emerge and before they grow out of hand. A number of CM-based workspace awareness tools (e.g., JAZZ [4], BSCW [6], Night Watch [5]) implement this strategy.

To date, no empirical evidence exists of workspace awareness tools being effective in reducing the number of conflicts in a development project. In this paper, we discuss the results of two formative experiments of our workspace awareness tool, Palantír [9]. Specifically, we evaluated Palantír by conducting two sets of pilot user experiments where subjects collaboratively solved a given set of programming tasks (some of which conflicted with each other) in three-person teams. In both experiments we observed that the experimental group, which used the full functionality of Palantír, was more effective in detecting conflicts earlier and produced a final product with fewer remaining unresolved indirect conflicts (all direct conflicts had to be resolved by both groups during check-in). Our results represent a beginning. While not statistically significant due to the size of the experiment, they suggest that workspace awareness supports effective coordination strategies.

The remainder of the paper is organized as follows. In Section 2, we discuss background information on workspace awareness and our tool, Palantír. Section 3 discusses our user experiments and their results. Section 4 presents our conclusions.

2. Background

Awareness is characterized as “an understanding of the activities of others, which provides a context for your own activity” [7]. Awareness as a concept can be applied to many different activities, but within the discipline of computer science it has been generally associated with the field of computer-supported cooperative work (CSCW). There, efforts have largely focused on the use of awareness in coordination in group activities (e.g., shared text editing, group decision making). In the more recent past, researchers have started investigating the concept of awareness in facilitating coordination in software development.

One of the primary problems involving coordination in software development is the lack of understanding of changes to artifacts by team members in their (remote) workspaces and – particularly – how such changes interrelate with ongoing changes in the local workspace. Workspace awareness aims to overcome this issue by continuously informing developers of remote activities, and using various analyses and annotations to indicate the nature and impact of those activities [8].

Palantír is an awareness tool that complements CM workspaces by collecting, distributing, organizing, and presenting information of workspace operations (both CM and editing operations). Particularly, it continuously informs developers of which artifacts are being concurrently changed by which developers, the size of those changes, and the impact of those changes on the local workspace. Palantír does this through visual cues that it peripherally embeds in the development environment. The primary goal is to warn developers of conflicts, as they emerge and grow, via awareness icons that draw the attention of the user, but do not drastically distract them from their primary (coding) task. Through additional visualizations, the developer can investigate any conflicts in detail. Further discussion of Palantír can be found in our previous work [9].

3. User Experiments

We designed two formative experiments to evaluate whether workspace awareness promotes users to self-coordinate in order to avoid conflicts. The objective of our experiments was to mimic development situations in which conflicts (both direct and indirect) would arise and to then observe if individuals noticed the conflicts and took action to resolve them.

The distributed nature of the activity allowed the experiments to be designed to test one subject at a time. Specifically, the experimental setup consisted of a subject collaboratively solving a given set of programming

(Java) tasks in a three-person team, where the other two team members were confederates – virtual entities controlled by the research personnel and responsible for introducing a given number of conflicts with the subject’s tasks. Subjects could reach their team members (confederates) via Instant Messaging. The use of confederates ensured consistency in the type, number, and timing of conflicts across experiments.

Subjects were undergraduate and graduate students in the Donald Bren School of Information and Computer Sciences at UC Irvine. All were very familiar with the development environment (Eclipse and CVS), but not with Palantír. Subjects were given a brief tutorial that was designed to ensure that subjects in the experimental group would not be biased to expect conflicts in the experiment. Subjects were asked to “think aloud” and their progress was observed by research personnel and recorded with screen capture software. Subjects were randomly assigned to be in the control or experimental group. In both experiments, subjects in the experimental group used all of the features of Palantír; subjects in the control group either used no Palantír, or a limited version (as discussed below for each experiment).

Experiment tasks. The software project contained 19 Java classes and approximately 500 lines of code. Subjects had to implement a set of 12 tasks, designed so that a subset of them conflicted with changes made by confederates. Of the 12 tasks assigned to the subject, 8 conflicted, 4 as *direct conflicts* (e.g., a confederate editing the same file) and 4 as *indirect conflicts* (e.g., a confederate deleting a method call that the subject is currently using). These conflicts were further divided into three categories: (1) conflicts introduced *before* the subject began a particular task, (2) conflicts introduced *during* a task, and (3) conflicts introduced *after* the subject had already completed their task. The order of the twelve tasks was allowed to fluctuate per subject, but the relative timing at which the conflicts were introduced by the confederate in response to subjects’ activities was exactly the same each time.

3.1. Experimental Findings

For each experiment, we analyzed: 1) detection and resolution rates of conflicts, 2) actions taken by subjects to self-coordinate, and 3) time-to-completion per task (including conflict resolution where applicable). Details of our hypothesis, research questions, findings, and threats to validity are provided in a technical report ([10]). In this paper, we share our primary results, illustrating that users actively utilize workspace awareness to keep track of emerging conflicts and employ various

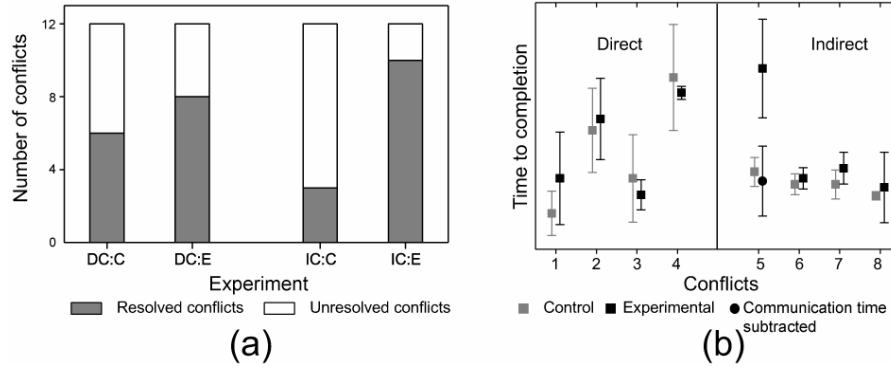


Figure 1. Results: (a) Conflict Resolution for Direct (DC) and Indirect Conflicts (IC) for Control (C) and Experimental (E) Groups; (b) Time-to-Completion.

proactive measures to avoid conflicts or resolve conflicts as soon as they are detected.

3.1.1. Experiment I. In this experiment, three subjects constituted the control group and three the experimental group. The experimental group used Palantír, which provided warnings of any direct and indirect conflicts. The control group used only Eclipse and CVS, with no awareness information. There were eight conflicts (four direct and four indirect) introduced per subject. Figure 1(a) presents the results of our analysis, as divided into four cases: direct and indirect conflicts (*DC* versus *IC*) for each group (Control versus *Experimental*). For each case, then, there were a total of 12 seeded conflicts in the experiment (4 conflicts per each of 3 subjects). We found no distinction between detection and resolution rates; subjects resolved all conflicts that they detected.

In the case of direct conflicts, the total number of conflicts resolved was similar in the control group and experimental group. However, subjects in the control group discovered these conflicts later, after they completed their task and tried to check in the artifacts. Subjects in the experimental group detected conflicts earlier, while their changes were in progress. An interesting point to note is that subjects rarely resolved conflicts concerning tasks for which they had completed and checked in their changes.

In the case of indirect conflicts, subjects in the experimental group discovered and resolved a larger number of conflicts than subjects in the control group. Merely one of the four seeded indirect conflicts was detected by the control group (by each subject), largely because the file that caused an indirect conflict also caused a direct conflict later in the experiment.

Time-to-completion. Figure 1(b) presents the time it took subjects to complete tasks involving a conflict. Times primarily indicate natural fluctuations caused by variations in technical aptitude. Conflict 5 (*IC*), though, exhibits a large difference, with the experimental group using more time (a three minute difference in the mean)

than the control group. This conflict was introduced while subjects were still completing the task at hand; as a result, subjects spent time communicating with the confederate. It has to be observed, though, that, although the experimental group took longer to complete this task, subjects proactively resolved the indirect conflict. The control group failed to detect the problem.

3.1.2. Experiment II. In this experiment our goal was to determine the effect of Palantír’s impact analysis in aiding subject’s in detecting indirect conflicts. Both of the groups used Palantír; the control group, however, was presented with notifications of direct conflicts only and subjects had to use their understanding of the software structure (they were provided with a UML design diagram) to identify indirect conflicts. The experimental group had explicit notifications of both direct and indirect conflicts.

Four subjects constituted the control group and four the experimental group. The total time to completion of the assignment was restricted to one hour. The average number of tasks that subjects completed within the time limit was eight. Our analysis, therefore, considers these first eight tasks only, which covered four conflicts (two direct and two indirect). Figure 2(a) presents our analysis, as split – once again – into four cases. Each case therefore had a total of 8 conflicts (2 conflicts per each of 4 subjects).

For direct conflicts, all of the measured performance indicators for both of the groups were the same since they used the same tool functionality. Similar to results in the first experiment, subjects did not resolve conflicts introduced after they had completed their task.

For indirect conflicts, subjects in the control group detected fewer indirect conflicts (only three out of eight were detected), despite being provided with information of the changes that caused the conflict and UML diagrams detailing dependency relations among artifacts. Subjects in the experimental group identified and resolved *all* the indirect conflicts. They used different

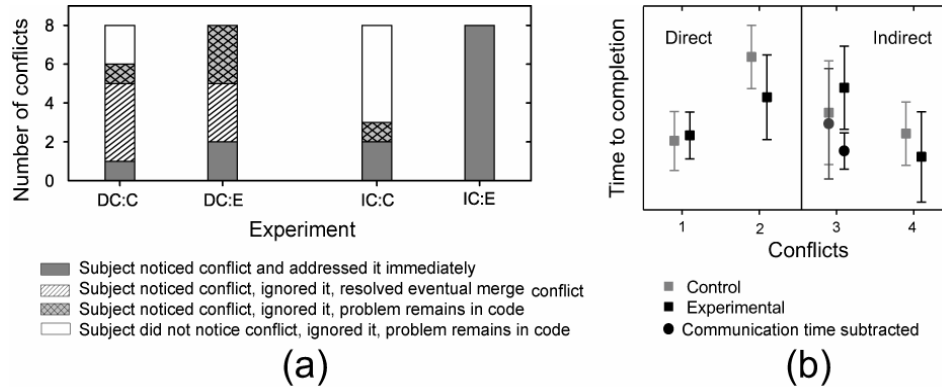


Figure 2. Results: (a) Conflict Resolution for Direct (DC) and Indirect Conflicts (IC) for Control (C) and Experimental (E) Groups; (b) Time-to-Completion.

strategies to avoid or resolve conflicts. Some skipped the task and came back to it later, some updated their workspace with the latest versions of the artifacts, some asked their team member to implement their tasks, and some coded the task with a place holder.

Time-to-completion: Figure 2(b) shows the time it took subjects to complete conflicting tasks. The times, again, highlight minor variations caused by differences in the technical aptitude of subjects. As in our first experiment, subjects in the experimental group took longer for one task with an indirect conflict, which involved the confederate introducing the conflict during the task. Of course, we note once again that subjects in the experimental group detected and resolved the conflict, while those in the control group did not.

4. Conclusions

Our experiments show that subjects indeed monitor awareness cues, especially for artifacts which they consider important. Moreover, on detecting conflicts, they take actions to self-coordinate and resolve the conflict. Subjects were comfortable in filtering out information (icons) that they felt were not important for their tasks. In our study, the experimental group was more successful in noting and resolving conflicts early, leading to an end product of higher quality (in terms of the number of indirect conflicts left unresolved in the project).

Our experiments are pilot in nature and do not offer statistically significant results due to the small number of subjects in each of the experiments. Nonetheless, our results provide a clear impetus to conduct further study to investigate the role of awareness in promoting self-coordination. Verbal comments from the subjects confirm the analyses presented here.

We are now conducting such experiments, particularly focusing on overcoming the problem of variances in technical aptitude by using text-based instead of coding-based tasks.

5. Acknowledgments

We thank Suzanne Schaefer and Gerald Bortis for their help in designing the experiments. Effort partially sponsored by NSF grants CCR-0093489, IIS-0205724, and IIS-0534775, as well as an IBM Eclipse Innovation grant and an IBM Technology Fellowship.

6. References

- [1] J. Estublier, et al., *Impact of Software Engineering Research on the Practice of Software Configuration Management*. TOSEM, 2005. 14(4): p. 1-48.
- [2] C.R.B. de Souza, et al. *Sometimes You Need to See Through Walls - A Field Study of Application Programming Interfaces*. CSCW. 2004. p. 63-71.
- [3] A. Sarma and A. van der Hoek. *A Conflict Detected Earlier is a Conflict Resolved Easier*. Workshop on Open Source Software Engineering. 2004. p. 82-86.
- [4] L.-T. Cheng, et al. *Jazzing up Eclipse with Collaborative Tools*. Eclipse Technology Exchange Workshop. 2003. p. 102-103.
- [5] C. O'Reilly, P. Morrow, and D. Bustard. *Improving Conflict Detection in Optimistic Concurrency Control Models*. Eleventh International Workshop on Software Configuration Management. 2003. p. 191-205.
- [6] W. Appelt. *WWW Based Collaboration with the BSCW System*. Conference on Current Trends in Theory and Informatics. 1999. p. 66-78.
- [7] P. Dourish and V. Bellotti. *Awareness and Coordination in Shared Workspaces*. CSCW. 1992. p. 107-114.
- [8] M.-A. Storey, D. Cubranic, and D.M. German. *On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and a Framework*. SOFTViz. 2005. p. 193-202.
- [9] A. Sarma, Z. Noroozi, and A. van der Hoek. *Palantir: Raising Awareness among Configuration Management Workspaces*. Twenty-fifth International Conference on Software Engineering. 2003. p. 444-454.
- [10] A. Sarma, A. Van der Hoek, and D. Redmiles, *A Comprehensive Evaluation of Workspace Awareness in Software Configuration Management Systems*. 2007, University of California, TR: UCI-ISR-07-2.