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
{asarma, andre, redmiles}@ics.uci.edu

Abstract

Workspace awareness has emerged as a new coordination paradigm in software configuration management systems, enabling the early detection of potential conflicts by providing developers with information of relevant, parallel activities. The focus of our work has been on detecting and mitigating both direct and indirect conflicts by unobtrusively 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 leads 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]. CM systems depend on repositories with well-defined access and synchronization protocols to facilitate multiple developers working on a common set of artifacts. In a typical scenario, developers check-out the required artifacts from the repository into private workspaces and, once their changes are complete, they synchronize their changes with the repository.

Private workspaces are essential in allowing developers to work without interference from others' changes, but have the negative effect of hiding knowledge of fellow team members' activities. As a result developers lose the context their work with respect to others' changes. Conflicts are thus detected later, only after developers have finished their changes and are ready to check-in. Furthermore, only *Direct Conflicts* – which arise due to changes to the same artifact – are detected by CM systems. *Indirect Conflicts* – which arise because of changes in one artifact affecting concurrent changes in another artifact – remain undetected until build testing or the deployment phase. Conflict resolution at such late stages is expensive and time consuming [2, 3].

One way to overcome this problem is to inform developers of ongoing activities that are 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. A number of CM based workspace awareness tools (e.g., JAZZ [4], Night Watch [5], BSCW [6]) implement this concept.

However, thus far, there exist no empirical evidences of such tools being effective in promoting self-coordination among developers to help reduce the incidence of conflicts in the project. In this paper, we discuss the results of formative evaluations of our workspace awareness tool, Palantír, in aiding the early detection and resolution of conflicts.

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 better in detecting conflicts earlier and produced a final product with fewer unresolved indirect conflicts (all direct conflicts had to be resolved during the check-in). This validates our hypothesis that workspace awareness promotes self-coordination and leads to the production of a higher quality end product in terms of the number of unresolved conflicts.

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 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 fellow team members' activities and how these changes affect the local workspace. Workspace awareness aims to overcome this problem by informing developers of which artifacts are concurrently being changed, which developers are making those changes, and the effects of those changes on the local workspace [8].

Palantír is a workspace awareness tool that complements CM workspaces by collecting, distributing, organizing, and presenting information of workspace operations (both CM as well as editing operations inside the development environment). Specifically, Palantír informs developers of which artifacts are being concurrently changed by which other developers, the size of the changes, and the impact of those changes on the local workspace through subtle cues peripherally embedded in the development environment. The primary goal is to warn developers of emerging conflicts (both direct and indirect) through awareness icons that draw the attention of the user, but not distract them from their primary task. Other complementary comprehensive visualizations aid an in-depth investigation of conflicts. Detailed discussion of Palantír can be found in our previous work [9].

3. User Experiments

We designed a set of formative evaluations to test whether workspace awareness promotes users to self-coordinate in order to avoid conflicts. The objective of our experiments was to mimic team software development where conflicts (both direct and indirect) would arise and observe individuals take action to resolve the conflicts with(out) the aid of workspace awareness.

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 or graduate students from the Computer Science department at UCI and were familiar with the development environment (Eclipse + CVS), but not with Palantír. Subjects were given a brief tutorial of functionalities of both these tools. The tutorials were designed to ensure that subjects in the experimental group were not biased to expect conflicts in the experiment. Subjects were asked to "think aloud" and their progress was observed by research personnel and recorded through screen capture software. Subjects were randomly assigned to the control or the experimental group. In both experiments, the experimental group used Palantír, while the conditions for the control group differed and are discussed separately for each experiment.

Experiment tasks. The software project contained nineteen Java classes and approximately 500 lines of code. As part of the experiment, subjects had to implement a set of twelve tasks, which were so designed that a subset of them conflicted with changes made by confederates. Of the twelve tasks assigned to the subject, eight conflicted, namely four *direct conflicts* (e.g., a confederate editing the same file as the subject) and four *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 entered the task, (2) conflicts introduced *during* the task (while the subject was performing the task), and (3) conflicts introduced *after* the subject had already completed the task. These conflicts were randomly seeded throughout the tasks.

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) to estimate the benefits of workspace awareness. A detailed discussion of our experiment questions, discussion of findings for each question, and threats to validity of our experiment are discussed elsewhere (see [10]). In this paper, we share our experiment results that distinctly show a trend where users actively utilize workspace awareness to keep track of emerging con-

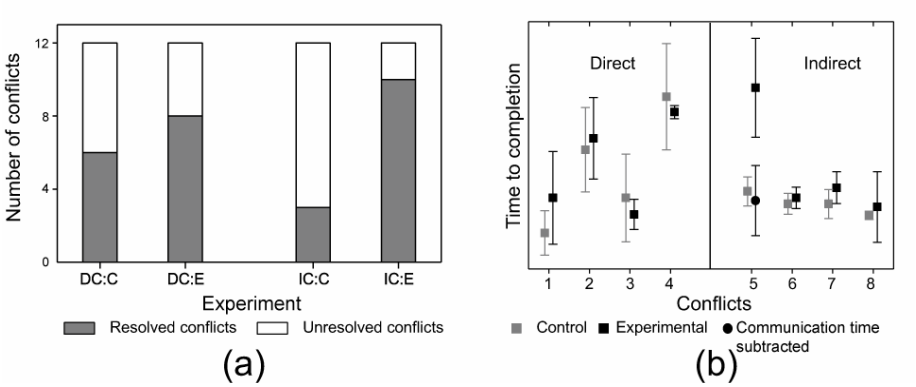


Figure 1. Experimental Results: (a) Conflict Detection and Resolution for Direct and Indirect Conflicts for Control and Experimental Groups; (b) Time-to-Completion.

flicts and employ various proactive measures to avoid conflicts or resolve conflicts as soon as they are detected.

3.1.1. Experiment I. We performed six experiments (*three* each for the control and the experimental group). The experimental group used Palantír, which provided them with warnings of potential direct and indirect conflicts, while 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) shows the results of our analysis, as divided into four cases: direct and indirect conflicts (*DC* versus *IC*) for each condition group (*C*ontrol versus *E*xperimental). For each case, then, there were 12 seeded conflicts (4 conflicts and 3 subjects). We found no distinction between detection and resolution rates; subjects resolved all the conflicts that they detected.

In the case of direct conflicts, the total numbers of conflicts resolved were similar across both groups. However, the control group discovered conflicts later, once they had completed their task and were trying to check-in; whereas the experimental group detected conflicts earlier, while subjects were still editing their tasks. An interesting point to note is that subjects rarely bothered to resolve conflicts in tasks once they were completed and checked-in. In the case of indirect conflicts, the experimental group discovered and resolved a larger number of conflicts than the control group. Only one out of three conflicts was detected by the control group, which was accidentally discovered because the file that caused an indirect conflict also caused a direct conflict later in the experiment.

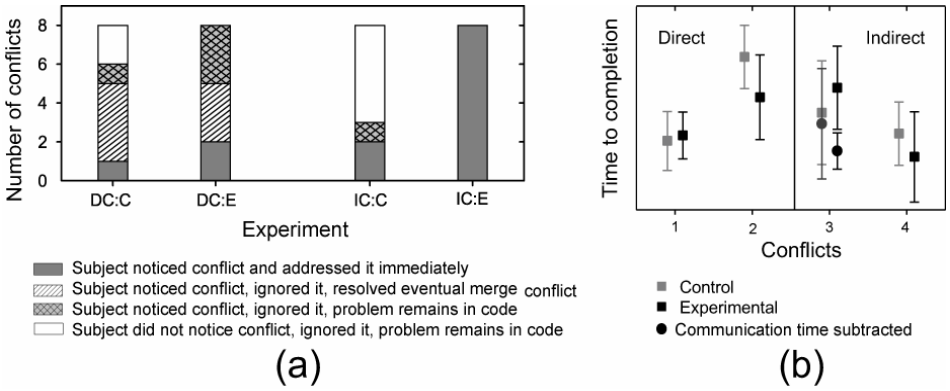
Time-to-completion. Figure 1(b) shows time-to-completion for the conflicting tasks. Times show natural fluctuations caused by variations in the technical aptitude of subjects. However, conflict 5 (IC) shows a marked difference, with the experimental group taking longer (three minute difference in the mean) than the control group. This anomaly was because the changes causing this conflict were still work-in-progress and the subjects spent time communicating with the confederate. The point to note is that, although the experimental group took longer to complete the task they proactively resolved the indirect conflict. The control group did not detect the problem in the code and never resolved it.

3.1.2. Experiment II. In this experiment our goal was to determine the effectiveness of impact analysis in aiding detection of indirect conflicts. Both conditions used Palantír. We provided the control group with only notifications of direct conflicts. Subjects had to use their understanding of the software structure (they were provided UML design diagrams) to manually identify indirect conflicts. The experimental group had explicit notifications of both direct and indirect conflicts.

In total, we performed eight experiments (four each for the control and 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, which included four conflicts (two direct and two indirect). Figure 2(a) presents our analysis, as split into four cases representing each kind of conflict for every condition. Each case therefore had a total of 8 conflicts (2 conflicts and 4 subjects).

For direct conflicts, all of the measured performance indicators for both the groups were the same since they used the same tool functionality. Similar to results form our previous experiments, subjects were not inclined to resolve conflicts once they had completed their task. For indirect conflicts, subjects had 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 strategies to avoid or resolve conflicts: they skipped the task and came back to it, updated their workspace,



asked their team member to implement their tasks, or coded the task with a place holder

Time-to-completion: Figure 2(b) shows time-to-completion for conflicting tasks. The times show minor variations caused by differences in the technical aptitude of subjects. Similar to our previous experiment set, the experimental group took longer to complete one task with an indirect conflict (conflict 4), which involved a work-in-progress task of the confederate. But the experimental group resolved the conflict, while the control group did not.

4. Conclusions

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

Although pilot in nature, our experiments clearly provide positive results and the impetus to conduct further studies that investigate the role of awareness in promoting self-coordination. Towards this goal, we are conducting a next set of experiments, which measures quantitative benefits and statistical proofs of workspace awareness promoting self-coordination and a better quality software (in terms of fewer conflicts left in the code) through. To overcome the problem of variances in the time-to-completion of tasks we will design the experiment to be text-based and not use the think aloud methodology.

5. Acknowledgments

We thank Suzanne Schaefer and Gerald Bortis for their help in designing the experiments. Effort partially funded 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. *Palantír: 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.