ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/4282130

A Comprehensive Evaluation of Workspace Awareness in Software Configuration Management Systems

Conference Paper · October 2007

DOI: 10.1109/VLHCC.2007.7 · Source: IEEE Xplore

CITATIONS
7

READS
41

3 authors:

Anita Sarma
Oregon State University
100 PUBLICATIONS **1,456** CITATIONS
SEE PROFILE

Andre van der Hoek
University of California, Irvine
213 PUBLICATIONS **5,238** CITATIONS
SEE PROFILE

David Redmiles
University of California, Irvine
202 PUBLICATIONS **4,195** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Information Foraging Theory View project

Merge Nature View project

All content following this page was uploaded by David Redmiles on 05 September 2015.

The user has requested enhancement of the downloaded file.

ISR Institute for Software Research
University of California, Irvine

A Comprehensive Evaluation of Workspace Awareness in Software Configuration Management Systems

Anita Sarma
University of California, Irvine
asarma@ics.uci.edu

André van der Hoek
University of California, Irvine
andre@ics.uci.edu

David F. Redmiles
University of California, Irvine
redmiles@ics.uci.edu

June 2007

ISR Technical Report # UCI-ISR-07-2

Institute for Software Research
ICS2 217
University of California, Irvine
Irvine, CA 92697-3455
www.isr.uci.edu

A Comprehensive Evaluation of Workspace Awareness in Software Configuration Management Systems

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

ISR Technical Report # UCI-ISR-07-2
June 2007

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 particular research in workspace awareness 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 a novel user experiment that we designed as a broad and formative evaluation of workspace awareness, specifically focusing on whether users detect conflicts as they arise and indeed act to mitigate the potential problems. Our results affirm that workspace awareness is an effective solution that promotes active self-coordination among users and furthermore leads to an improved end-product in terms of its quality.

A Comprehensive Evaluation of Workspace Awareness in Software Configuration Management Systems

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

ISR Technical Report # UCI-ISR-07-2
June 2007

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 particular research in workspace awareness 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 a novel user experiment that we designed as a broad and formative evaluation of workspace awareness, specifically focusing on whether users detect conflicts as they arise and indeed act to mitigate the potential problems. Our results affirm that workspace awareness is an effective solution that promotes active self-coordination among users and furthermore leads to an improved end-product in terms of its quality.

1. Introduction

Software development is considered to be "multi-person construction of multi-version software" [1]. As in any team effort, coordination is an integral part of software development. However, coordinating software development activities is not an easy task, as it typically involves complex interdependencies among large numbers of artifacts, developers, and tightly-coupled tasks. In addition, time pressures, parallel development, and distributed teams intensify these challenges [2, 3].

While there are numerous different coordination solutions available for use, Configuration Management (CM) systems have become one of the most popular and widely adopted tools in the software industry [4]. CM systems handle the situation of multiple developers working together on a common set of artifacts by providing a central repository with well-defined access and synchronization protocols. In a typical CM scenario, developers check-out the required artifacts from the central repository into their 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 they have the negative effect of hiding knowledge of fellow team members' activities As a result of which developers cannot place their work in the context of others' changes. Conflicts are, thus, detected only after developers finish 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 even after the deployment phase. Conflict resolution at such late stages is expensive and time consuming [3, 5].

One way to overcome this problem is to inform developers of other 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 and self-coordinate their actions. This concept has been implemented through workspace awareness tools that enhance CM workspaces with awareness information [6].

However, thus far, there exists no concrete evidence of such tools being effective in reducing the incidence of conflicts in the project or promoting self-coordination among developers. In this paper, we discuss the results of our evaluations of Palantír in aiding

the early detection of conflicts. Palantír is a workspace awareness tool that 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 [5].

We evaluated Palantír by conducting two sets of 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. 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 tools. Section 3 briefly describes Palantír. Section 4 discusses our user experiments and their results. Section 5 presents our lessons learned from the experiments with conclusions in Section 6.

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 [6, 8].

Researchers have built many workspace awareness tools [6]. BSCW [9] and Jazz [8] are workspace awareness tools that provide information of which developer is editing which artifact and the state of the artifacts to the developers' local workspace. While BSCW provides a web-based interface, Jazz is an Eclipse-integrated collaborative development. In a similar fashion, FASTDash [10] and the War Room Command Console [11] publicly display the set of artifacts checked-out in private workspaces and highlight artifacts that are opened for viewing or being concurrently edited. All of these tools primarily support the early detection of direct conflicts. To detect potential indirect conflicts, developers have to manually interpret the information of concurrent changes to artifacts along with their knowledge of the software structure.

Automatic detection of indirect conflicts is less straightforward and typically requires program analysis. In addition to Palantír, only one other workspace awareness tool performs semantic analysis to identify the impact of concurrent changes. TUKAN [12] performs program analysis on the copy of the software stored in the repository to determine which artifacts are semantically related and creates a semantic network of artifacts. It then uses this network to determine if ongoing changes to artifacts in local workspaces affect other artifacts in the graph, warning users with icons if so.

There have been numerous studies on software development practices which have identified that developers spend a large portion of their time understanding team members' activities. As per our knowledge, there is no empirical evidence proving the effectiveness of CM based workspace awareness tools.

3. Palantír

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). Currently, Palantír is built as an Eclipse plug-in for CVS and Subversion CM systems. An in-depth discussion of Palantír can be found in our previous work [5]. Briefly, Palantír Workspace Wrappers collect and emit events regarding relevant workspace activities. These events are stored and distributed by a Palantír Server, which also supports bootstrapping any new workspaces that developers may open to perform their work. The Palantír Client pulls, stores, and organizes the events, which are unobtrusively displayed to users.

Figure 1 shows a visualization provided by Palantír, where we see a developer's (Pete's) view of his local workspace. Palantír has altered the Eclipse interface in two distinct places. Annotations in the package explorer view provide subtle awareness cues (inset of Figure 1) and a new Eclipse view, the Impact View, provides further details of changes causing indirect conflicts (bottom of Figure 1). Palantír annotates resources in the package explorer view with both graphical and textual items. In terms of graphics, two small

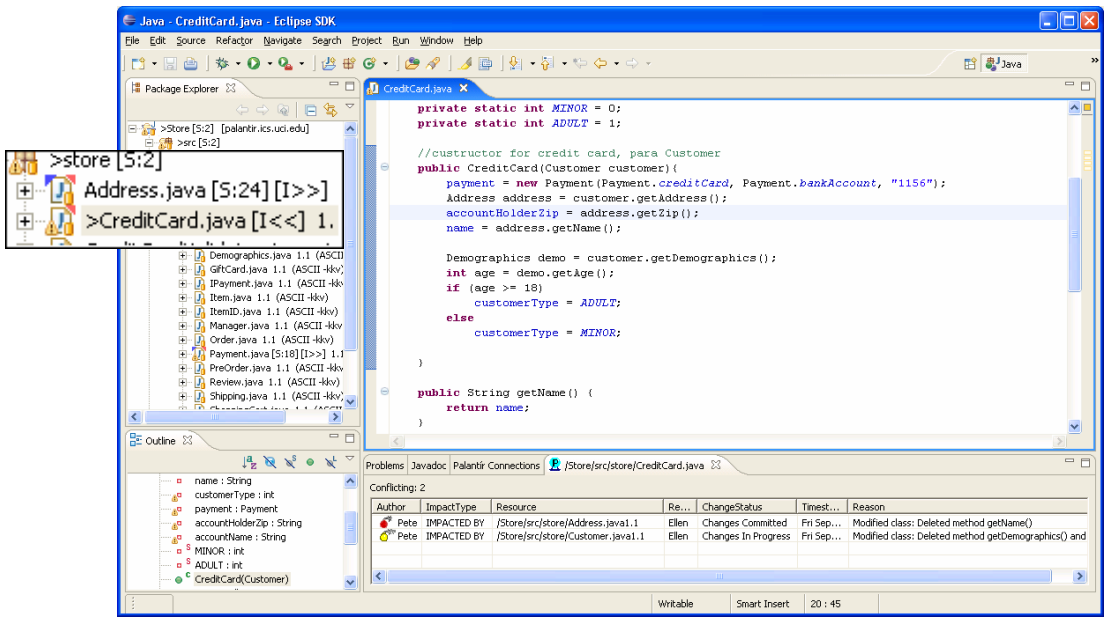

Figure 1. Palantír Visualization of Indirect Conflicts, With a Call-out of the Package Explorer.

triangles indicate parallel changes to artifacts. The first one, blue in color, may appear in the top left corner. This triangle, visible on *Address.java*, *Customer.java*, and *Payment.java*, indicates that there are ongoing parallel changes to the artifact. The larger the triangle, the greater are the changes. A small textual annotation, to the right of the filename, details the size of these changes and is based on the relative lines of code changed. In our case, *Address* has been changed by 24%.

The second triangle, red in color, may appear in the top right corner of each icon denoting an artifact. This triangle is present on four artifacts: *Address.java*, *Customer.java*, *Payment.java*, and *CreditCard.java*, and indicates the presence of an indirect conflict. A small textual annotation, to the right of the filename, helps the developer distinguish whether the artifact is one causing an indirect conflict (in which case it is labeled with [I>>] to denote "outgoing impact"), is one that is affected by an indirect conflict (in which case it is labeled with [I<<] to denote "incoming impact"), or is one whose changes affect changes in other artifacts and are affected by changes in other artifacts (in which case both textual annotations are present).

The package explorer view is designed to be non-obtrusive and only provides the information that is necessary to draw the user's attention. Further details about the indirect conflict are presented through the Palantír Impact View, when an artifact experiencing an indirect conflict is selected in the package explorer.

In our example, Pete is investigating indirect conflicts affecting *CreditCard.java*. It turns out that another developer (Ellen) has changed *Address.java* and deleted a method that he began using. Since this change is already committed to the CM repository and is a predictable build conflict, this indirect conflict is annotated with a mini icon of a "red bomb". The second line details a similar problem: deletion of a method from *Customer.java* that Pete is currently using in his implementation. This change is annotated with a mini icon of a "yellow bomb", because the changes are still in Ellen's workspace, but would create a build conflict in the future should she commit them.

Such information of ongoing changes allows Pete to place his work in the context of Ellen's changes and he can take proactive measures, such as contacting Ellen to discuss the logic and her timeline for completion of her changes. Based on their conversation, Pete may, for instance, decide to work on *CreditCard.java* later after Ellen has completed all her changes.

4. User Experiments

Palantír is a workspace awareness tool that informs developers of ongoing changes, giving them the opportunity to self-coordinate and produce software with fewer conflicts. We conducted two sets of user experiments to answer the following questions:

Q1. Do subjects notice awareness icons and understand their significance?

Q2. Do subjects initiate coordination actions on noticing the icons?

Q3. Do subjects in the experimental group detect conflicts earlier than subjects in the control group?

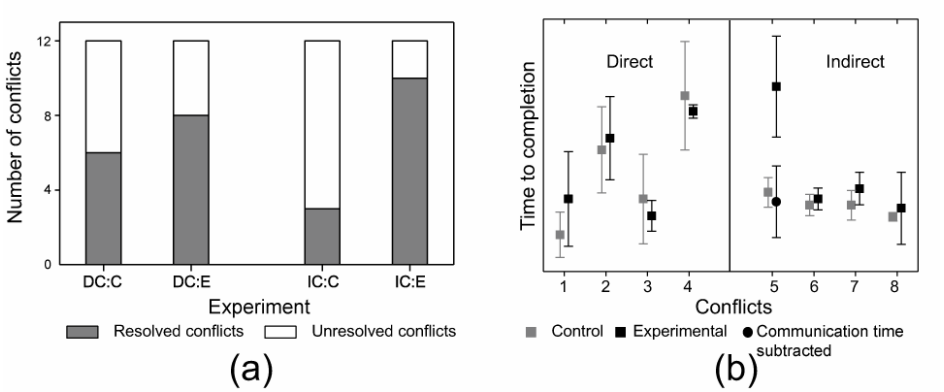

Figure 2. Experimental Results: (a) Conflict Resolution for Direct and Indirect Conflicts for Control and Experimental Groups; (b) Time-to-Completion.

Q4. Does the experimental group resolve a larger number of conflicts successfully?

Q5. Is time-to-completion of the assignment lower for the experimental group?

4.1. Experimental Design

The overall goal of the design was to mimic team software development where conflicts would arise and individuals take action to resolve them. However, 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. 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 feature request that translated into a set of twelve tasks. The tasks were designed so that some of the changes by the subject conflicted with those of the confederates'. Of the twelve tasks assigned to the subject, eight conflicted, namely four direct conflicts (DC) and four indirect conflicts (IC). 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. Each confederate was responsible for four conflicts, as well as benign changes that did not affect the subject.

4.2. 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). Our experiment results show that the experimental group was better in resolving a larger number of conflicts, especially indirect conflicts in the project.

4.2.1. Experiment I. We performed six individual 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.

The primary goal of our analysis was to determine whether subjects detected (and resolved) potential conflicts and the time-to-completion for each task. There were eight conflicts (four direct and four indirect) introduced per subject. Figure 2(a) shows the results of our analysis, as divided into four cases direct and indirect conflicts (DC versus IC) for each condition group (Control versus Experimental). For each case, then, there were 12 seeded conflicts (4 conflicts and 3 sub-

jects). We found no distinction between detection and resolution rates; subjects resolved all the conflicts that they detected.

Case DC:C: When subjects were not provided with any information of parallel activities they became aware of direct conflicts when attempting to check-in as notified via a merge conflict. We noticed that after facing a merge conflict, subjects were more cautious and sometimes contacted (IM) their team members to ask about the files that they were changing to create a context for their changes. One such IM communication is "*...what sets of methods are you implementing?...Have you looked at [artifact name] yet? Just wondering should I go about implementing….*" Direct conflicts that were introduced after the subject had finished their task (1 conflict) or changes that were still work-in progress by the confederates could not be detected and therefore were left unresolved (1 conflict).

Case DC:E: The majority of the subjects detected and resolved potential direct conflicts before they had even completed their changes. Of the four conflicts, one conflict was completely avoided (conflict introduced before the subject started the task) and two detected and resolved during the task. Additionally, one subject resolved the conflict that was introduced after the subject had finished their task and another that was still work-in-progress (IM conversations). The other two subjects did not resolve conflicts that were introduced after their task or which were work-in-progress.

Case IC:C: Subjects did not detect the majority of indirect conflicts; only one conflict was detected by all subjects. This was because the same file that caused an indirect conflict also caused a direct conflict. When users updated their workspaces to resolve a merge conflict, they detected the indirect conflict due to a local build failure. The other three indirect conflicts remained undetected in the project.

Case IC:E: Subjects identified and resolved all indirect conflicts, except one subject who did not detect two conflicts. One reason for the subject not detecting one of the conflicts was that the warning icon on the top right corner of the artifact was hidden behind a "compilation error" icon in Eclipse, which the subject did not resolve.

Time-to-completion: Figure 2(b) shows time-to-completion for tasks that were designed to conflict. The times show natural fluctuations caused by variations in the technical aptitude of subjects. However, for conflict 5 (IC), we note 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. Literature points out that the resolution of conflicts later in the development stage is much more expensive and time consuming [13]. Since subjects in our experiments were not required to conduct integration testing, we cannot conclusively determine how much longer the control group would have taken to resolve the problem. If we subtract the time that the experimental group spent in communications, we note that both groups become more or less equal.

Discussion: The experiment questions can be answered as follows:

Q1. Subjects in the experimental group noticed awareness icons with a rate of 75% and understood their significance. We found that subjects were not inclined to investigate (resolve) direct conflicts that were introduced after they had completed their tasks. This was primarily because in CM systems, the developer who checks in second is responsible for resolving conflicts.

Q2. Subjects in the experimental group *always* initiated coordination actions once they noticed the icons.

Q3. Subjects in the experimental group detected conflicts earlier (70.8% times, the other times they either ignored the "*after"* direct conflict or they missed the warning icons) compared to subjects in the control group (only 4% times). Only one subject in the control group detected a direct conflict early, because they were extra cautious and, before starting every task, they queried the repository to ensure that they had the latest version. Although the subject therefore detected the conflict early, they had to spend extra effort in repeatedly querying the repository.

Q4. Subjects in the experimental group resolved a larger number of conflicts (75%) than the control group (37.5%). Since all direct conflicts were resolved at the latest during check-in time, the main difference lies in the detection and resolution of indirect conflicts. Indirect conflicts are usually harder to detect and prove more expensive to resolve in real life settings. Early detection of such conflicts is, therefore, particularly desirable.

Q5. Times to completion for subjects in both groups were similar. However, the experimental group produced a better quality end product with fewer unresolved conflicts.

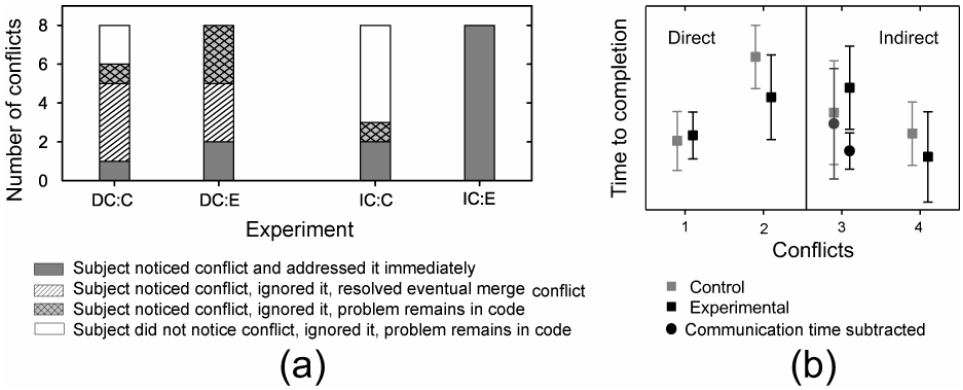

Figure 3. Experimental Results: (a) Conflict Detection and Resolution for Direct and Indirect Conflicts for Control and Experimental Groups; (b) Time-to-Completion.

4.2.2. 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. The control group subjects had to use their understanding of the software structure (they were provided UML design diagrams) to manually identify indirect conflicts. The experimental group was provided with information of concurrent changes to artifacts along with explicit warnings of potential indirect conflicts.

In total, we performed eight individual 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). In particular we focused on the detection and resolution rates of conflicts and the time-to-completion of tasks. Figure 3(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).

Case DC:C: The majority of the subjects noticed the warning icons; six out of eight conflicts were detected. However, subjects did not immediately realize that they could have avoided the conflict by communicating with their team members (only one conflict was avoided). "*I had noticed the blue icons, but I was in the train of thoughts… but after I ran into trouble, it provided me an incentive to talk to my team member and monitor the icons.*" Similar to our previous experiments, we found that subjects mainly monitored for conflicts before starting a task and before committing their changes. Subjects largely ignored "*after*" conflict warnings.

Case DC:E: Subjects showed similar results as in the previous case since both groups had access to the same functionality of Palantír (warnings of potential direct conflicts). The difference was that all potential conflicts were detected, of which two were avoided as two subjects immediately understood the significance of the problem and updated their workspace and/or communicated with their team members. One of them commented "*…before I started working on it, Palantír tells me that someone is changing it, so I went and checked, saw that everything is there, so cool, task completed and no conflicts*". Subjects did not resolve conflicts that were introduced *after* they completed their tasks.

Case IC:C: We found that subjects had difficulty detecting indirect conflicts (only three out of eight were detected), despite providing them with information of the changes that caused the conflict and UML diagrams detailing dependency relations among artifacts. Only one subject could detect and resolve both indirect conflicts, primarily because they proceeded cautiously and continuously monitored concurrent editing warnings and frequently updated their workspace "*…because I am traumatized, I had past problems with committing things without updating, so I always synchronize my workspace before and after I finish a task*". It is important to note that, though a step in the right direction, the workspace synchronizations in and of themselves were not sufficient. Frequent workspace updates meant that subjects had to carefully examine and update their code in response to parallel changes.

Case IC:E: Subjects identified and resolved all the indirect conflicts and used both the Package Explorer extension as well as the Impact View. A subject said: "*…the icons, those were very helpful to determine like it was an impact…I found it really useful because I could sort of anticipate that there would be conflict just by looking, and …I could know what I needed to do, so I could have time to prepare, or like I did, I contacted him [confederate] directly to ask him what was happening at that moment.*" Subjects 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 3(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. It is important to note that, although the experimental group took longer to complete the task, they produced a higher quality code in terms of unresolved conflicts.

Discussion: For direct conflicts, all of the measured performance indicators for both the groups were the same (within the statistical uncertainty). For indirect conflicts we answer our experiment questions as:

Q1. Subjects noticed awareness clues with a rate of 100% and understood their significance. This high success rate may be attributed to the fact that subjects had already experienced instances of direct conflicts and were more cautious.

Q2. Subjects initiated coordination actions on noticing the icons in 100% of the cases. This corroborates our previous experiment observations that when subjects detect potential conflicts that affect them, they *always* take measures to self-coordinate.

Q3. Subjects in the experimental group detected conflicts earlier (100%) than subjects in the control group (37.5%). This demonstrates the fact that understanding the software structure and manually placing concurrent changes in context to identify indirect conflicts is a difficult task. The few conflicts that were detected were because some subjects were extremely cautious and frequently updated their workspaces (before embarking on a task and before committing their changes).

Q4. Subjects in the experimental group resolved a larger number of conflicts (100%) than the control group (37.5%). This once more proves the fact that, although it is possible to use information on direct conflicts and the software structure to look for indirect conflicts, it is difficult to do so.

Q5. Time to completion of tasks for both groups was more or less similar. The difference being that the experimental group had resolved all the indirect conflicts while the control group did not.

5. Lessons Learned

Our experiments qualitatively prove that the awareness of parallel activities (especially information of potential conflicts) promotes self-coordination among developers and leads to an end product of higher quality. Next, we will conduct a larger set of experiments to statistically validate our hypothesis, where we will use a similar experimental design with minor, but important modifications (see Sections 5.2 and 5.3). We specifically plan to use text-based assignments to reduce variances in time-to-completion per task due to differences in technical aptitude [14].

5.1. Experiment Model Limitations

As is the case with any controlled experiment, our experiments were performed in a semi-realistic setting. Our subjects were undergraduate and graduate students with limited real-life development experience and were asked complete a given set of tasks in a limited time. Subjects had no prior experience with Palantír and had to learn its functionality as they performed their tasks. In industrial settings, it is possible that subjects will be more concerned about the quality of the code and behave accordingly.

We believe that the complexity of real life projects will make the use of Palantír (or other workspace awareness tools) considerably more beneficial than that demonstrated in the experiments. Our experiments involved a small project (nineteen classes), easy tasks, and confederates readily available for conflict resolution. All of these factors are more difficult to deal with in real life situations. However, in very large projects the Palantír user interface may face scalability issues when a large number of artifacts are changed frequently. We still need to evaluate Palantír is such situations, even though we have taken especial care to make Palantír scalable and unobtrusive [5].

Finally, our “time to completion” data is inconclusive because of the large variance in the subjects’ performance due to their differences in technical capabilities and the fact that subjects were not required to perform integration testing. The next set of experiments will use text-based assignments and subjects will be required to remove all inconsistencies from the project at the end of the experiment.

5.2. Experimental Design Changes

We found that, although students had used CVS before, they had limited experience in resolving conflicts through CVS. We also found that subjects had difficulty interpreting the Impact View. To overcome these

issues, we will conduct a detailed tutorial on CVS usage and Palantír for our next experiments.

In our current experiments, we randomly assigned subjects to the control or the experimental groups. However, we found that subjects had marked differences in their technical backgrounds. In the future, we will use stratified random assignment to ensure that both groups have subjects with comparable technical backgrounds [15].

Finally, based on our exit interviews we found that many subjects were uncomfortable with the "think aloud" process. Specifically, international students faced an extra cognitive load to verbalize their thought process in English. Our next experiments will not require the think aloud process.

5.3. Interface Design Changes

Our observations and interviews from the experiments prompted us to make changes to the Package Explorer extension. First, we will add textual information about the identity of the developer editing the artifact and the status of the change (work-in-progress or checked-in) in the package explorer extension. Currently, Palantír only provides textual annotations about the cumulative size of parallel changes to the artifact. Users have to use either the CVS resource history view or another Palantír visualization to find the specifics, which users found cumbersome.

Second, we will change Palantír to only annotate artifacts with incoming impact. Information about the artifact that is causing the indirect conflict can then be obtained from the Palantír Impact View. In the current version, artifacts causing an impact, as well as artifact that are affected are annotated with graphical icons and text. Subjects found two annotations for a single conflict confusing and had difficulty navigating the view.

6. Conclusions

Workspace awareness is a coordination strategy in software engineering that relies on the presentation of subtle visual cues embedded in the development editor to inform developers of relevant parallel changes in other workspaces. The goal is to enable developers to become aware of parallel changes as they occur, so that developers can place their work in the context of others' and self-coordinate.

We conducted two sets of user experiments to evaluate the efficacy of Palantír, our workspace awareness tool, in helping developers identify and resolve conflicts. Our results clearly show that subjects monitored the visual cues displayed, especially for artifacts which they considered important. The majority of them then took actions to self-coordinate. In fact, the experimental groups (where Palantír provided warnings of indirect conflicts) resolved a much larger percentage of indirect conflicts than the control groups (with no warnings of indirect conflicts). Further, we found that subjects were quite comfortable in filtering out information (icons) that they felt as not important for their tasks. Our results qualitatively prove that workspace awareness prompts users to self-coordinate and lead to an end product of higher quality (in terms of the number of indirect conflicts left unresolved in the project). We are currently conducting a larger set of experiments to statistically validate our findings.

7. Acknowledgments

We thank Suzanne Schaefer and Gerald Bortis for their help in designing and conducting the experiments. Effort partially funded by the National Science Foundation under grant numbers CCR-0093489, IIS-0205724, and IIS-0534775, as well as an IBM Eclipse Innovation grant and an IBM Technology Fellowship.

8. References

[1] D.L. Parnas, *Some Software Engineering Principles*. Infotech State of the Art Report on Structured Analysis and Design, Infotech International, 1978: pp 10.

[2] I. Vessey and A.P. Sravanapudi, *CASE Tools as Collaborative Support Technologies*, ACM CACM. 1995. p. 83-95.

[3] 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.

[4] J. Estublier, et al., *Impact of Software Engineering Research on the Practice of Software Configuration Management*. ACM TOSEM, 2005. 14(4): p. 1-48.

[5] A. Sarma, Z. Noroozi, and A. van der Hoek. *Palantír: Raising Awareness among Configuration Management Workspaces*. ICSE. 2003. p. 444-454.

[6] M.-A.D. 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*. ACM Symp. on Software Visualization. 2005. p.193-202.

[7] P. Dourish and V. Bellotti. *Awareness and Coordination in Shared Workspaces*. CSCW. 1992. p. 107-114.

[8] L.-T. Cheng, et al., *Building Collaboration into IDEs. Edit ->Compile ->Run ->Debug ->Collaborate?* ACM Queue. 2003. p. 40-50.

[9] W. Appelt. *WWW Based Collaboration with the BSCW System*. Conference on Current Trends in Theory and Informatics. 1999. p. 66-78.

[10] J. Biehl, et al. FASTDash: *A Visual Dashboard for Fostering Awareness in Software Teams*. Computer/Human Interaction (CHI 07). 2007. p. (to appear).

[11] C. O'Reilly, D. Bustard, and P. Morrow. *The War Room Command Console: Shared Visualizations for Inclusive Team Coordination*. ACM symposium on Software visualization. 2005. p. 57-65.

[12] T. Schümmer and J.M. Haake. *Supporting Distributed Software Development by Modes of Collaboration*. ECSCW. 2001. p. 79-98.
[13] F.P. Brooks Jr., *The Mythical Man-Month*. Datamation, 1974. 20(12): p. 44-52.
[14] A.J. Jacko and A. Sears, *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*. 1 ed. 2002: pp. 1296.
[15] W.R. Shadish, T.D. Cook, and D.T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. 1 ed. 2001: pp. 623.

View publication stats