How to Evaluate a Conflict Minimizing Task Scheduler through a User Study

Bakhtiar Khan Kasi and Anita Sarma
Computer Science and Engineering Department
University of Nebraska-Lincoln
Lincoln, NE, USA
{bkasi, asarma}@cse.unl.edu

***Abstract*—Workspace awareness tools facilitate coordination among developers in a team by informing them of emerging conflicts due to parallel development. Several such tools have been introduced recently. However, evaluating such (collaborative) tools through user studies is nontrivial because it depends on the group dynamics and their development behavior. In this paper, we present the challenges in evaluating a collaboration tool geared towards minimizing conflicts by scheduling (independent) development tasks. We present the research questions that a user evaluation should answer along with the foreseen challenges in answering these questions. We would like to use the workshop to exchange opinions and feedback to refine the design of our user study and start a conversation on the challenges and methods for evaluating a collaborative development tools.**

***Index Terms*—Evaluation, user studies, task scheduling.**

I. Introduction

Workspace awareness tools [1]–[3] have become popular in the recent past as they are designed to facilitate development by introducing new ways to improve task coordination, enabling developers to identify and address a variety of software conflicts and coordination problems. However, empirically evaluating the benefits of these tools is nontrivial and only few of them have been evaluated from the perspective of user studies e.g. Palantír [1], FASTDash [2] and CollabVS [4].

While evaluating a tool by deploying it in industry settings is the gold standard, not all researchers have the required resources to do so. Understandably, companies are hesitant to experiment with new prototypes, especially those developed externally. Another challenge with evaluating tools through deployment is that, to be successful, collaborative tools require the buy-in of the entire team, which is especially difficult.

Because of these challenges and the fact that deployment is not the best strategy for performing feasibility study with initial versions of prototypes, researchers often first test prototypes in controlled lab settings. Results of which can later be generalized to industrial environments.

There are significant challenges in evaluating collaborative tools in lab settings too [5]. The success of a collaboration tool (even something as simple as an eclipse plugin embedded in a developers work environment) depends among others, on developers' skills, their experiences, and coordination with fellow developers. These factors need to be simulated appropriately, so as to make the experiment realistic.

To create such realistic experimental settings, the following requirements need to be met. First, the prototype should be available within the typical work context that a developer is likely to use (e.g., a plugin within the Eclipse Platform). Second, the tool should be easily usable, without distracting developer from the main task at hand (e.g., coding or debugging). Third, the experiment participants need to be experienced enough not only in the environment (e.g., Eclipse) but also in the domain (e.g., the programming language in which the tasks are created) and the type of task (e.g., debugging vs. coding). Fourth, experimental tasks have to be carefully formulated with realistic complexity and rich design. Moreover, tasks needs to simple enough to be completed in a short period of time within a session.

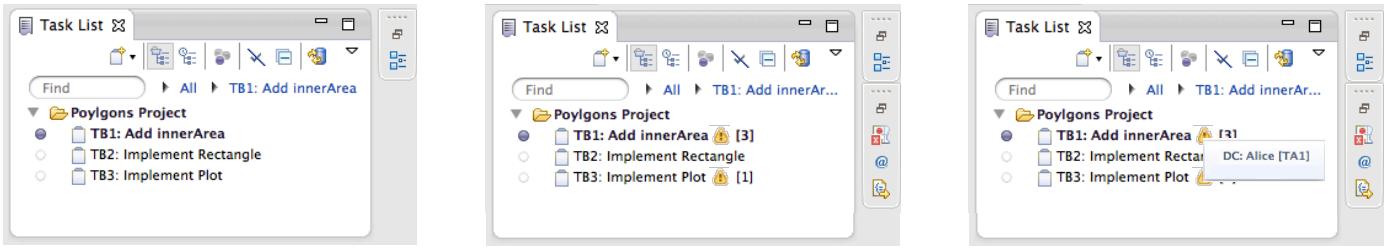
Finally, in case of collaborative tools the team settings and tasks for team members have to be simulated. For example, in evaluating a collaboration tool, the evaluation scenario should include users who are in different rooms, performing their individual tasks and interacting with each other via the tool or other communication protocols (e.g., chat sessions or emails). If the goal is to study how an individual will use and react to the collaboration prototype, a confederate study design might be needed to simulate these scenarios so that each individual is evaluated in isolation to control for variances that might arise because of variations in group interactions.

In this paper we identify challenges for evaluating collaboration tools especially one that is geared towards minimizing conflicts in teams. We then present the research questions that we would like to answer when evaluating our prototype, Cassandra, which is an optimized task scheduling, awareness tool [6]. We provide here an initial plan for a user study to evaluate Cassandra, however, our main goal is to use the workshop for seeking feedback on the design of our user study.

The rest of the paper is organized as follows. Section II discusses our prototype Cassandra. In Section III, we present challenges and difficulties we faced in performing our previous artifact-centered evaluation of Cassandra. We present evaluation questions and our proposed evaluation in Section IV. We conclude with our goals for the workshop in Section V.

II. Cassandra

Cassandra is a novel task scheduling system that aims to minimize conflicts by recommending task orders that restrict dependent tasks or tasks that share common files from being



(a) (b) (c)

Fig. 1. A mockup of Cassandra's user interface. (a) The leftmost panel shows the task list with tasks in a user specified order. (b) The middle panel shows tasks with conflicts as indicated by the warning icons and a recommended task ordering which is shown as [*n*]. (c) The rightmost panel shows further details about a conflict that is visible via a mouse hover event

concurrently edited [6]. Cassandra enables a proactive conflict minimization technique by identifying potential conflicts between tasks if they are performed in parallel, and consequently recommends conflict-free tasks to developers. Cassandra optimizes the solution space by comparing the task orders (if more than one exists) to ordering desired by the developer (developer preference) and selects the order that best matches the developers' preferences.

We present the core functionality of Cassandra as follows:

Task context identification: Cassandra allows users to create and arrange tasks as per their preference. The context generator component is implemented in Eclipse development environment as an extension to Mylyn's plugin [7] and is responsible for tracking the development context of a task (the files that are being currently edited or marked for future edits by a developer). Currently, the user is required to identify the task context a priori. In the future, an initial list of files that are likely to be edited per task will be automatically generated through mining past tasks and the files changed for that task. The user would then be expected to refine this list through the interface.

Task scheduling: Cassandra identifies and formalizes task dependencies, task precedence, and developer preferences into constraints. The constraints are then evaluated using the Z3 SMT solver [8] to identify "satisfiable" solutions. If a conflict free solution is found, then it is optimized to match developer preferences to the extent possible. If no solution exists, then constraints are progressively relaxed until a solution is found.

Constraint reevaluation: Cassandra reevaluates the constraint space periodically to ensure that the satisfiability of the solution is up-to-date. If new constraints are found the task ordering can be updated for future tasks of a developer. The fact that constraints are reevaluated periodically and that (future) task order recommendations can be updated makes the solution robust as it accounts for cases of non-precise predictions. Cassandra reevaluates the constraints when a user has completed the task and is checking in.

User interface: The user interface for Cassandra is implemented as an extension to the *task-list view* of Mylyn (see Fig. 1). Users interact with the UI to prioritize tasks based on their preferences, view the recommended task order and conflict information for the tasks in their workspace.

Let us consider an example where Bob is interacting with the Eclipse IDE at the beginning of his workday. Fig. 1 illustrates three different views of the task-list available to Bob. Using the UI, he first creates three tasks and then orders them (e.g. TB1, TB2 and TB3 in Fig. 1 (a)) as per his preference. Let us assume that he also identifies the files that are going to be changed for each task. Once the tasks have been identified and organized, Cassandra (in the background) evaluates the constraints between Bob's tasks and other tasks in the project, identifying tasks that will face conflicts (annotated with warning symbols as shown in Fig. 1 (b)). Cassandra also shows the recommended task sequence *n* (e.g., TB3…[1] means that TB3 should be performed first).

The user can also view additional information on the conflict by hovering over the warning symbol (as shown in (Fig 1. c)). The conflict information provides details about which tasks by which developers are potentially conflicting. In our example (Fig. 1 (c)) Bob's task TB1 has a "Direct Conflict" (same files are likely to be edited in parallel) with Alice's task TA1. Note that there could be more than one conflict among tasks.

III. Evaluation Challenges

To test the feasibility of evaluating and solving constraints among developer tasks in a project, we first conducted an artifact-only study of Cassandra [6]. Here we briefly discuss the study and the challenges we faced, as they have a bearing on our future studies including user experiments.

Archival data analysis: We performed an empirical analysis conducted on four open source projects hosted on GitHub (Perl[1], Storm[2], Jenkins[3], and Voldemort[4]); through historical data gathered based on change set activity. We first quantified the number of conflicts and their types (merge conflicts, build failures, test failures) and determined the resolution efforts for each conflict. We then used this data for evaluating the task-scheduling component of Cassandra, to show the feasibility of the scheduling technique. That is, we investigated change sets per project on a weekly basis and identified alternate task orders that would have avoided conflicts in that time frame.

While this archival process showed the existence of possible non-conflicting task orders for the project, we could not retroactively integrate the change sets in the new task sequence. This was because the change sets in the historical data, had

[1]http://www.perl.org
[2]http://storm-project.net
[3]http://jenkins-ci.org
[4]http://project-voldemort.com

inherent functional dependencies and retrospectively changing their order when integrating them into the master repository led to a different set of conflicts. This limitation shows that in the absence of additional information about functional dependencies among change sets, retrospectively reordering and integrating them is infeasible.

Simulation based analysis: Since the projects that we evaluated were open source (OSS) projects with limited amount of parallel development, our data set only had limited number of conflicts (e.g., the project Storm had 975 change sets, out of which there were 17 merge conflicts, 9 build failures, and 13 test failures). To stress test our scheduling algorithm and technique, we therefore simulated data to explore situations with a higher number of constraints and conflicts.

The simulation data was generated by mutating one of the four open source projects (Storm). We used this data to test Cassandra's efficiency with high numbers of constraints. However, it should be noted that the simulation generated mutants of existing conflicts and cannot be generalized to all conflicts.

These experiments have shown the feasibility of a task scheduler to identify optimum non-conflicting task orders. However, these evaluations did not evaluate the UI, or whether users would find the tool usable and act upon the recommendations, or whether users would take the time and effort to create/refine the task context.

Our objective for future studies is to conduct an experimental evaluation of Cassandra as an integrated system. Because our solution inherently relies on both the technological functionality of Cassandra as well the human responses to the information provided by the tool, our evaluation should focus on this interplay [9].

IV. Experiment Design

We plan to conduct user study within a university-based environment to evaluate the effectiveness and efficiency of Cassandra. The experiment has to simulate a team setting where participants work in a team and where some of the tasks are meant to conflict. We plan to compare, how using Cassandra (in the Experimental group) allows fewer conflicts to occur than not using it (Control group).

In order to appropriately evaluate Cassandra we would like to answer the following research questions:

RQ1: *Does using Cassandra allow users to avoid conflicts?*

Here we would like to compare the number of conflicts faced by the Experimental group to those faced by the Control group. When seeding the conflicts in the tasks, we will use the distribution of conflicts that we have found in the OSS projects. We have to identify the number of conflicts and their types that can be seeded in the experiment setting. For example, we can only have (as many) tasks that the user can complete in a single session (2 hours maximum) and not all tasks have to be conflicting. We also have to ensure that the resolution of the conflicts is not too complex, to avoid biasing the experiment in our favor.

RQ2: *How well do users understand the task recommendation and how often do they follow it?*

This is a key question to answer. If users are able to understand the recommendations and faithfully follow it then they will not face any conflicts. We are interested in evaluating the satisfaction and level of trust that users have on Cassandra. However, given that this is an experimental setting, a threat to validity is that users will be more likely to follow the tool recommendations. To make the experiment more realistic we might have to include some false positives.

RQ3: *What is the consequence of violating the recommended task sequence?*

While Cassandra recommends the optimum task orders, it is possible that a user may not follow the recommendation and follow their preferred order. We would like to investigate, how often a user violates the task sequence and under which circumstances. We could design a think-aloud experiment so that we can gain insight into their actions at the different stages of the experiment. More information on a particular behavior pattern may be obtained through exit interviews.

RQ4: *Does Cassandra affect the time-to-completion of tasks by proactively identifying conflicts and avoiding them?*

Conflict resolution requires coordination and delays the time to complete a task, we would therefore like to see how much of the time can be saved if conflicts could be avoided by using Cassandra. However, note that Cassandra requires upfront developer effort in identifying the files that are going to be edited per task. We plan to evaluate how this upfront effort compares to (saved) conflict resolution effort.

We will compare the time taken by participants in Control and Experimental group to complete their tasks. As mentioned earlier, in order to avoid biasing the experiment, we need to ensure that conflict resolution is not overly expensive. This exercise will also help quantify the differences in resolution times for different types of conflict. Note, that it is possible that participants in the Experimental group will face conflicts if they do not follow the recommended task order.

A. Evaluation Challenges

When answering the above research questions within an experimental study, we have to appropriately design the study to control for the following threats to validity:

Individual differences: Individual difference among participants can significantly affect the outcome of an experiment. For example, the time it takes a user to complete a programming task or resolve a conflict can vary widely depending on their experiences. There are several options to reduce the effects of this problem. For example, we can perform a within-subjects study design. However, given the fact that we would like a user to complete several programming task it is unlikely that we will be able to have a user participating in both treatment groups without causing experimenter fatigue. Another option is to use stratified random selection; that is users are grouped into different strata based on their background from which users are selected at random for each treatment.

It is possible that our participants do not have the experience of working in teams or the expertise in merging changes. Further, resolving a build or test failure is nontrivial. In order to minimize the impact of individual differences in development

expertise we might include both programming and non-programming tasks. We will investigate the use of a text-based task assignment, where dependencies across text files can be simulated [8].

Designing the control group: Designing the experimental setup for the Control group is probably the most challenging task in tool evaluation. As mentioned earlier, it may be unjustified to compare the performance of subjects in the Experimental group (that allows conflicts to be avoided) to those solely relying on their own skills and needing to resolve conflicts in the Control group. In the absence of a similar scheduling tool (that can be used as common baseline) the best we can do is provide the Mylyn interface to the Control group.

Designing appropriate tasks: Designing tasks that are appropriate for the evaluation is a major challenge. We want to maintain a certain degree of complexity among tasks so they are neither too trivial nor overly complex for the participants. However, the actual tasks should also require some effort from the participants so that the users primarily focus on the tasks and not the scheduling part. The complexity of the conflicts and their resolution similarly cannot be too complex.

Confederate design: A key technique that Cassandra uses to identify conflict-free tasks is to identify constraints across pairs of tasks so as to identify a sequence of tasks for each developer that can be performed independently. In our example, Bob's Task TB1 conflicts with Alice's TA1, so they both cannot be performed in parallel and the tool recommends Bob to work on TB3 instead. However, if Alice were to work on TA2, which does not conflict with TB1, Bob would be free to work on that task (TB1). Therefore, the order in which each team member performs his/her task impacts the task ordering proposed by Cassandra. Maintaining the same order of tasks (and therefore a similar set of conflicts) across different experiment runs will therefore be not possible, unless we exert control over the sequence in which tasks are performed.

We are planning on evaluating one user at a time and simulate a team and their actions through the use of confederates, research personnel acting as virtual team member [9]. The confederates will work as team members with their identity hidden from other participants to mitigate any bias that a confederate may develop. They will monitor the task ordering chosen by the user and appropriately select their own tasks – leading to conflicting situations or vice versa.

The use of confederates will also be needed for the Control group, where conflicts are supposed to occur if a user follows a given task order. If we use real users to function as confederates, the rate at which each performs a task and their (own) task preference might impact the incidence of conflicts (or a lack thereof). To be able to compare both the groups, we need to exert control over these parameters.

Think-aloud process: An important challenge in user experiments is the ability to capture a collaboration situation in its entirety. For example being able to provide a holistic view of the situation or insight into what makes a participant behave in a certain way and what are the factors that influence certain decisions [10]. While performing a think-aloud study protocol helps us to gain insight into participant behavior and motives, it outlaws the analysis of times-to-completions as thinking aloud needs cognitive effort and can hold up participant's performance. We plan to perform a small set of think-aloud experiments and then rely on exit interviews to gain insight into participant behavior.

V. Workshop Goals

Here we have presented our evaluation questions and the challenges in evaluating our prototype, designed to minimize conflicts in a team development scenario. The key challenge is creating an experimental design where users in the Experimental group can avoid conflicts by following the recommendations through Cassandra, but not to heavily bias the experiment towards its favor.

We would like to take the opportunity of the workshop to get feedback and refine our design of the user study. We would also like to identify additional risks in our study that we can mitigate, before performing the study.

Acknowledgment

The work is partially funded by NSF grants: IIS-1253786 and CCF-1110916

References

[1] A. Sarma, Z. Noroozi, and A. van der Hoek, "Palantír: Raising Awareness among Configuration Management Workspaces," *25th International Conference on Software Engineering*, pp. 444–454, 2003.

[2] J. Biehl, M. Czerwinski, G. Smith, and G. Robertson, "FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams," *SIGCHI Conference on Human Factors in Computing Systems*, pp. 1313–1322, 2007.

[3] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Proactive detection of collaboration conflicts," *19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 168–178, 2011.

[4] P. Dewan and R. Hegde, "Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development," *2007 Tenth European Conference on Computer-Supported Cooperative Work*, pp. 159–178, 2007.

[5] O. Badreddin and T. C. Lethbridge, "Combining experiments and grounded theory to evaluate a research prototype: Lessons from the umple model-oriented programming technology," *User Evaluation for Software Engineering Researchers*, pp. 1–4, 2012.

[6] B. K. Kasi and A. Sarma, "Cassandra: Proactive Conflict Minimization through Optimized Task Scheduling," *35th International Conference on Software Engineering*, in-press See draft at http://interaction.unl.edu/cassandra/resources

[7] M. Kersten and G. C. Murphy, "Mylar: A Degree-of-Interest Model for IDEs," *4th International Conference on Aspect-Oriented Software Development*, pp. 159–168, 2005.

[8] L. De Moura and N. Bjørner, "Z3: An Efficient SMT Solver," *Theory and Practice of Software*, pp. 337–340, 2008.

[9] A. Sarma, D. Redmiles, and A. van der Hoek, "Palantír: Early Detection of Development Conflicts Arising from Parallel Code Changes," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 889 –908, Aug. 2011.

[10] J. Schenk, "Evaluating awareness information in distributed collaborative editing by software-engineers," *User Evaluation for Software Engineering Researchers,* pp. 35 –38, 2012.