Timely and Efficient Facilitation of Coordination of Software Developers' Activities

Kelly Blincoe
Advisor: Giuseppe Valetto
Computer Science Department
Drexel University
3141 Chestnut Street, Philadelphia, PA. 19104, USA
https://www.cs.drexel.edu/~kac358/
kelly.blincoe@drexel.edu

***Abstract*— Work dependencies often exist between the developers of a software project. These dependencies frequently result in a need for coordination between the involved developers. However, developers are not always aware of these Coordination Requirements. Current methods which detect the need to coordinate rely on information which is available only after development work has been completed. This does not enable developers to act on their coordination needs. Furthermore, even if developers were aware of all Coordination Requirements, they likely would be overwhelmed by the large number and would not be able to effectively follow up directly with the developers involved in each dependent task. I will investigate a more timely method to determine Coordination Requirements in a software development team as they emerge and how to focus the developers' attention on the most crucial ones. Further, I hope to prove that direct inter-personal communication is not always necessary to fulfill these requirements and gain insight on how we can develop tools that encourage cheaper forms of coordination.**

Keywords- Awareness, Proximity, Management, Coordination Requirements, Socio-Technical, Task Context, Tools.

I. TECHNICAL PROBLEM

In large software projects, multiple developers must work together and concurrently. This requires a division of work which often results in dependencies between tasks. Software engineering pioneers, such as Parnas [18] and Brooks [3], recognized the importance of efficiently managing work dependencies to manage the coordination overhead arising within a development team. Although recent work on search-based optimization techniques has shown that optimization of project scheduling can reduce coordination overheard [23], work dependencies cannot be eliminated and those dependencies often result in Coordination Requirements (CRs) among team members. When developers either remain unaware or do not obtain timely awareness of the work dependencies that exist and the coordination that is required to fulfill these dependencies, there is a potential for problems that may affect the efficiency of the development process or the quality of the software product.

Although CR detection techniques exist, they do not yet detect CRs in a timely fashion or assess the relative importance or criticality of CRs. Such a detection method is required to effectively raise developers' awareness [9] of their coordination needs and empower them to act upon those needs. We also need to investigate what coordination strategies can be most efficient in fulfilling CRs once they are recognized.

II. RELATED WORK

Cataldo et al. [6] introduced a framework to detect and quantify CRs between pairs of software developers by identifying the technical dependencies between software artifacts modified during assigned tasks. Empirical evidence suggests that when coordination activities focus on the identified CRs productivity is likely to improve [5,6]. This has led to the concept of Socio-Technical Congruence (STC) [4,6] which states that when coordination is focused between the team members with identified CRs we can obtain benefits for the software project.

Taking advantage of those benefits requires the real-time detection of CRs, but current CR detection methods are not yet timely. CRs are usually identified by examining the artifact commits made by developers in the project's source control repository. Commit data is typically available only after the majority of development work for a task has been completed. Also, commit data is typically incomplete for two reasons. First, only a subset of developers may be granted commit privileges, so the commit history may portray inaccurate author information. Second, for each file committed to a source code repository, a developer may have consulted several other files. Knowledge of this source code reference behavior is inaccessible from commit records.

Tools such as Ariadne [7], EEL [17], Tesseract [20], and Codebook [1] employ abstractions similar to CRs to provide awareness to developers. They all depend on establishing technical dependencies among artifacts using commit data in the source code repository. They then use these dependencies to compute relationships between developers. Therefore, these tools are also unable to provide timely notifications that can raise the developers' awareness of their coordination needs. Other tools attempt to leverage live workspace information. For example, Palantír uses notifications to keep a developer abreast with what happens in her colleagues' workspaces [19]. Like the other tools, Palantír makes use of information from the configuration management system. However, instead of looking at commit data, it looks at the artifacts in each developer's workspace and their state. It then compares them to the state of the "master copy" for the same artifacts maintained in the configuration management repository. It notifies developers of changes occurring to the artifacts they have in their own workspace. While these notifications are timely, they only regard direct conflicts on the same artifact which are a narrow subset of CRs. Another

tool, CollabVS, also notifies developers of artifact conflicts, and it captures additional conflicts by considering a subset of syntactical dependencies between artifacts [8]. However, it does not measure the "strength" or importance of CRs.

Until CRs can be detected and quantified in real-time, they will not become an actionable concept for managing coordination in software projects.

III. Research Questions

A. Is timely Coordination Requirement detection possible?

Current algorithms rely on identifying the technical dependencies manifested by the software artifacts to detect CRs. As mentioned, the drawback of this approach is that this data is only available after all, or at least a substantial amount of, the development work has been completed. Is there a more timely way to accurately determine and compute CRs? If so, can such a method make CRs actionable by detecting them as they emerge during collaborative software development activities?

B. Are all Coordination Requirements created equal?

Current methods for detecting Coordination Requirements assume that all work dependencies may require coordination, but is this necessarily true? It is possible that certain types of work do not require coordination even when technical dependencies between tasks exist. If certain work dependencies do not require coordination, we could ignore them when computing CRs in order to avoid excessive coordination overhead.

Although some ways to rank CRs by importance [10,16] have been presented, current CR detection algorithms do not differentiate between less or more intense CRs and do not predicate on different kinds of CRs. Understanding what characteristics of interdependent software development tasks or the artifacts involved in them, may influence the need for coordination remains an open problem.

C. Are cheaper forms of coordination as effective?

Explicit coordination, through either synchronous or asynchronous communication acts, such as email, chat, face-to-face meetings, or phone calls, can be expensive. Another important form of coordination is implicit coordination. Implicit coordination happens by obtaining information about a task by watching another developer as they complete that task or by examining the effect of their work, such as changes to artifacts [11]. In software development, implicit coordination occurs via *stigmergy* when enough information is contained within a software artifact or its associated meta-data to enable a new developer to pick up that software artifact and complete a task that is already underway or accomplish a new dependent task without resorting to explicit coordination [12].

Recent team-oriented development environments, such as IBM Jazz [13], have introduced features like tags and dashboards to help promote forms of implicit coordination. It is currently unclear how effective these implicit coordination facilities can be. Treude and Storey found that the use of tags was quickly adopted by an industrial development team [21]. However, there has been limited research on the usefulness of tags in software development. I plan to research whether or not tagging software artifacts – as a stigmergic medium - is helpful in aiding coordination. I will also look at how we can integrate tagging and CR detection to provide awareness of, and support to, coordination needs.

IV. Proposed Methods and Evaluation

A. Is timely Coordination Requirement detection possible?

1) Method

I propose an alternative approach to the current reliance on technical dependencies for CR detection. My approach examines the similarity of artifact working sets as they are incrementally constructed throughout the course of developers' work. Working sets can be obtained by recording developers' actions on artifacts as they occur. The Mylyn framework (formerly Mylar) [14,15] is one existing tool that performs this recording function. I have developed a measure, called proximity, which looks at artifact consultation and modification activities captured by the Mylyn framework to weigh the overlap which exists between pairs of working sets associated to developers or tasks. Through an empirical study, I have found that proximity is indicative of the need to coordinate [2].

The proximity algorithm considers all actions recorded for each artifact in each working set in order to apply a weight to that artifact's proximity contribution. Weights are applied based on the type of overlap where the most weight is given when an artifact is edited in both working sets and the least amount of weight is given when an artifact is simply consulted in both working sets. Artifacts that do not appear in both working sets will not receive any weight. The weights, based on weights Mylyn itself uses for its degree-of-interest model [18], are: 1 for edit overlap, .79 for mixed overlap (edit in one working set and selection in the other), and .59 for selection overlap.

The algorithm then computes the ratio of actual to potential overlap. Actual overlap considers the intersection of the two working sets while potential overlap considers the union of the two working sets. Potential overlap represents the maximum possible proximity score had there been perfect overlap between the two sets of actions. The proximity measure is the ratio between the actual overlap and the potential overlap. An example of the proximity calculation is shown in Figure 1 [2].

2) Evaluation

To evaluate the proximity measure, I performed an empirical study that compared proximity to the CRs detected by the Cataldo et al. [6] CR detection algorithm. I found that higher values of proximity correlate with the likelihood of a CR. I also found that proximity has high levels of precision and recall when matched to the CRs (which for this evaluation I assumed as ground truth). I also examined the cases when the CRs and proximity scores do not align, and all cases examined turned out to be false positives or negatives of the traditional CR detection method. More importantly, several of those cases highlight drawbacks of that method's reliance

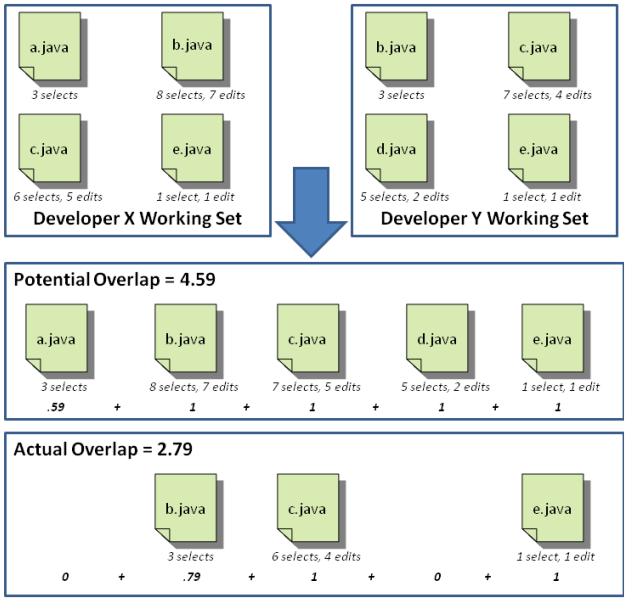

Figure 1. Proximity Algorithm [2].

on post-mortem information and dependency conceptualizations [2].

To evaluate the timeliness of the proximity measure, I obtained the time when the first contribution to the proximity score would have occurred by examining the timestamps for the overlapping events recorded in the working set pairs. I then compared the first proximity event with both the first day of concurrent work by that pair and the day in which the first CR is identified for the same pairs. I found that proximity significantly improves the timeliness of CR detection. For example, in one data set, concurrent work intervals last 31.4 days on average. Proximity is detected on average 6.2 days after parallel work begins while the CR detection method would not detect the CR until 25.2 days on average after the start of concurrent work [2].

A prototype of a tool which implements the proximity algorithm is currently being developed [24]. Future work includes analysis of how the tool and proximity measure supports timely and accurate CR detection.

B. *Are all Coordination Requirements created equal?*

1) *Method*

As the next step in the proposed research agenda, I plan to perform an empirical analysis to determine if certain classes of tasks or artifact manipulations are less likely to induce coordination requirements than others even when technical dependencies or working set overlaps exist between tasks. I will gather data on task characteristics, such as the type of task and the types of artifacts involved. The task type (new feature, feature modification, or feature removal) will be determined by looking at the number of lines of code added/changed/deleted and the labels put on the task itself. I am also interested in the types of artifacts involved in the tasks such as high-level interface consultation or low-level class modification.

2) *Evaluation*

I aim to observe if the characteristics of tasks/artifact sets and their communication levels correlate to the amount of 1) reverted changes, 2) reopened tasks, and 3) low task productivity. Such characteristics could indicate that more coordination was necessary showing that certain types of CRs are candidates for increased/prioritized coordination. On the other hand, characteristics of task pairs which indicate a CR yet show no reverted changes and high productivity could indicate these types of CRs do not actually require coordination.

C. *Are cheaper forms of coordination as effective?*

1) *Method*

I intend to focus on whether or not tags placed on software artifacts are helpful in aiding coordination since tagging is a prevalent form of implicit coordination embedded within existing collaborative development environments. I will look for task pairs which have coordination needs based on the actionable socio-technical model developed through research questions A and B. I will then perform an empirical study on those task pairs looking at the use of tags, communication levels, and productivity measures.

2) *Evaluation*

An empirical study will be performed which looks to see if the amount of communication found in task pairs correlates to the amount of artifact tagging involved in the tasks. I hypothesize that large amounts of tagging will correlate to lower levels of communication indicating that tagging serves as a form of implicit coordination. To determine the effectiveness of tagging, I will then look to see if task pairs with no traces of coordination other than the use of tags have similar productivity levels as those task pairs which do have traces of explicit coordination proving the effectiveness of tagging as form of coordination as compared to explicit forms of coordination.

Since tagging is a simple tool whose usage can greatly vary, I also plan to observe how tags are employed. I will look at what tagging styles are most conducive to productivity benefits and how to leverage them for coordination and CR awareness.

V. Expected Contributions

A socio-technical model constructed using developers' actions on artifacts as they occur will provide an actionable and "live" view of CRs as they are established. This will allow for prioritization of project governance actions aimed at the resolution of CRs that improve productivity the most [10]. As an alternative, changes could be made to the design or the team structure to eliminate CRs [22] lowering the coordination overhead of the project. Further, a determination of task characteristics which do not require coordination even when technical dependencies between tasks exist will allow developers to focus their attention on tasks where coordination is truly needed. Tools, such as a coordination recommendation engine, can then be developed

to make developers aware of their coordination requirements in real time while avoiding a large number of false positives.

Even more helpful than a recommendation engine which always encourages explicit coordination is one which would foster implicit coordination mechanisms. More knowledge about the effectiveness of tags, a simple type of implicit coordination, can help make progress in this direction. Such a recommendation engine could point developers to review tags on artifacts of interest or even automatically disseminate those tags rather than encouraging developers to seek a discussion with other team mates.

VI. Current Progress

I have begun my research in determining if timely Coordination Requirement detection is possible. I have put forward the proximity measure which calculates CRs based on the overlap between two working sets. My initial work validates the proximity idea by means of an empirical study on an open source project. I found that proximity provides an early indication of CRs and overcomes known drawbacks in current CR detection methods. I am now about to begin my research work on the remaining research questions put forth in this research abstract.

VII. List of Publications

Blincoe, K., Valetto, G. and Goggins, S. 2012. Proximity: a Measure to Quantify the Need for Developers' Coordination. In *Proceedings of the International Conference on Computer Supported Cooperative Work (CSCW 2012)*.

Blincoe, K. and Valetto, G. 2010. Implicit Coordination in Software Development. In *Proceedings of the International Conference on Global Software Engineering* (ICGSE 2010).

References

[1] Begel, A., Phang, K.Y., and Zimmerman, T. 2010 Codebook: Discovering and Exploiting Relationships in Software Repositories. Proc. ICSE 2010.

[2] Blincoe, K., Valetto, G. and Goggins, S. 2012. Proximity: a Measure to Quantify the Need for Developers' Coordination. Proc CSCW 2012.

[3] Brooks, F.P. 1995. The Mythical Man-Month: Essays on Software Engineering. Addison Wesley. Reading, MA.

[4] Cataldo, M, Herbsleb, J., Carley, K. 2008. Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development Productivity. Proc. ESEM 2008.

[5] Cataldo, M., Mockus, A., Roberts, J.A and Herbsleb, J.D. 2009. *Software Dependencies, Work Dependencies and Their Impact on Failures*. IEEE Transactions on Software Engineering, Vol. 35, No. 6, pp. 864-878

[6] Cataldo, M., Wagstrom, P.A., Herbsleb, J.D., and Carley, K.M. 2006. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. Proc. CSCW 2006.

[7] de Souza, C.R., Quirk, S., Trainer, E., and Redmiles, D.F. 2007. Supporting collaborative software development through the visualization of socio-technical dependencies. Proc. of the 2007 international ACM conference on Supporting group work. 147-156.

[8] Dewan, P. and R. Hegde. 2007. Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development. Proc. E-CSCW 2007. p. 159-178.

[9] Dourish, P., and Bellotti, V. Awareness and Coordination in Shared Workspaces. Proc. CSCW 1992: p. 107-114.

[10] Ehrlich, K., Helander, M., Valetto, G., Davies, S., and Williams, C. 2008. An analysis of congruence gaps and their effect on distributed software development. Proc. STC 2008.

[11] Gutwin, C. Penner, R. and Schneider, K., Group Awareness in Distributed Software Development, ACM conference on Computer Supported Cooperative Work, Chicago, Illinois, USA, 2004, pp. 72-81.

[12] Heylighen, - Why is open access development so successful? Stigmergic organization and the economics of information. Open Source Jahrbuch 2007.

[13] IBM Rational Jazz. http://www-01.ibm.com/software/rational/jazz/features/

[14] Kersten, M. and Murphy, G.C. 2005. Mylar: a degree-of-interest model for IDEs. Proc. AOSE 2005, 159-168.

[15] Kersten, M. and Murphy, G.C. 2006. Using task context to improve programmer productivity. Proc. FSE 2006.

[16] Kwan, I. and Damian, D. Extending Socio-technical Congruence with Awareness Relationships. Proc SSE 2011.

[17] Minto, S. and Murphy, G.C. 2007. Recommending emergent teams. Proc. MSR 2007.

[18] Parnas, D.L. 1972. On the criteria to be used in decomposing systems into modules. Communications of the ACM. 15, 12, 1058.

[19] Sarma, A., Noroozi, Z., and van der Hoek, A. Palantír: raising awareness among configuration management workspaces. Proc. ICSE 2003.

[20] Sarma, A., Maccherone,L., Wagstrom, P., and Herbsleb, J. 2009. Tesseract: Interactive visual exploration of socio-technical relationships in software development. Proc ICSE 2009, 23-33.

[21] Treude and Storey. How tagging helps bridge the gap between social and technical aspects in software development. In Proc. ICSE 2009.

[22] Valetto, G., Chulani, S., and Williams, C. 2008. Balancing the value and risk of socio-technical congruence. Proc. STC 2008.

[23] Di Penta, M., Harman, M., Antoniol, G., and Qureshi, F. 2007. The Effect of Communication Overhead on Software Maintenance Project Staffing: a Search-Based Approach. In Proc. ICSM 2007.

[24] Borici, A., Schröter, A., Damian, D., Blincoe, K., and Valetto, G. ProxiScientia: Toward Real-Time Visualization of Task and Developer Dependencies in Collaborating Software Development Teams. *Under Review*, 2012.