Timely Detection of Coordination Requirements to Support Collaboration among Software Developers

Kelly Blincoe
Computer Science Department
Drexel University
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. I have investigated a more timely method to determine Coordination Requirements in a software development team as they emerge.**

I. Problem and Motivation

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 [14] and Brooks [2], recognized the importance of efficiently managing work dependencies to manage the coordination overhead arising within a development team.

Work dependencies often result in Coordination Requirements (CRs) among team members. Developers, however, often remain unaware of the work dependencies that exist and the coordination that is required to fulfill these dependencies. When developers do not follow up on existing Coordination Requirements, there is a potential for problems that may affect the efficiency of the development process or the quality of the software product [4,10,17].

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 [8] of their coordination needs and empower them to act upon those needs. This would make the CR concept actionable and open coordination strategies that can fulfill CRs efficiently.

II. Background and Related Work

Cataldo et al. [5] 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 [4,5,17]. This has led to the concept of Socio-Technical Congruence (STC) [3,5] 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 timely detection of CRs, but with current detection methods, CRs are not an actionable concept. CRs are usually identified by examining the task's 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 incomplete for two reasons. First, the commit history may portray inaccurate author information due to limited commit privileges. Second, for each file committed to a source code repository, a developer may have consulted several other files pertinent to her work. Knowledge of this source code reference behavior is inaccessible from commit records.

Several tools, such as Ariadne [6], EEL [13], and Tesseract [16], employ conceptualizations of CRs that rely upon commit records to establish technical dependencies among tasks in order to provide developers with coordination awareness. Therefore, these tools cannot provide timely notifications of CRs. 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 [15]. Palantír also 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 compares them to the state of the "master copy" for the same artifacts maintained in the configuration management repository. It then notifies developers of ongoing changes occurring to the artifacts they have in their own workspace. While these notifications are timely, they only regard direct same-artifact conflicts, 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 [7]. However, it does not rank the "strength" or importance of the detected CRs and does not account for CRs that do not arise from other types of dependencies.

III. Approach

I have proposed an alternative approach to the current reliance on technical dependencies for CR detection which is timely and turns CRs into an actionable concept for managing coordination in software projects. My approach examines the similarity of artifact *working sets* as they are constructed during developers' tasks. Working sets can be obtained by recording developers' actions on artifacts as they occur. The Mylyn framework [11,12] is one tool that performs this recording function. I have developed the *proximity measure* which looks at artifact consultation and modification activities captured by Mylyn and weighs the overlap which exists between the working sets associated to pairs of developers or tasks. I have found that proximity is indicative of the need to coordinate [1].

The proximity algorithm considers all actions recorded for each artifact in each working set in order to apply a numeric 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, which are directly based on weights Mylyn itself uses for its degree-of-interest model [13], are shown in Figure 1 which also provides an example of the proximity computation process [1]. 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 actual overlap and potential overlap.

IV. RESULTS

To evaluate the accuracy of the proximity measure, I performed in [1] an empirical study that compared proximity to the CRs detected by the algorithm proposed by Cataldo et al. [5]. I found that higher values of proximity correlate with the likelihood of a CR. In our main data set, a Spearman correlation of 0.69 was found with a p-value of $2.4e^{-11}$. I also found that proximity has high levels of precision and recall when matched to the CRs conceptualized in Cataldo et al. (which in this case are assumed to be ground truth). I also found out that all cases when the CRs and proximity scores do not align, turned out to be either false positives or negatives of the traditional CR detection method. More importantly, several of those cases highlight drawbacks of that method's reliance on post-mortem information and dependency conceptualizations [1].

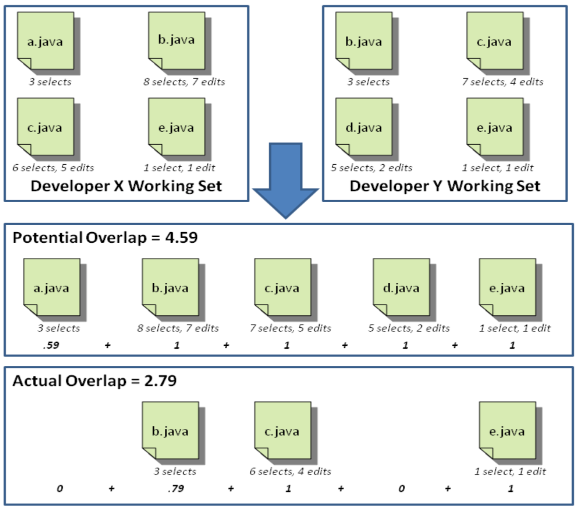

Figure 1. Proximity algorithm [1].

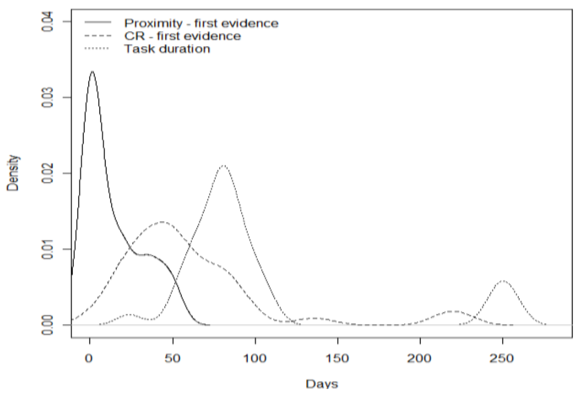

Figure 2. Timeliness Probability Density [1]

To evaluate the timeliness of the proximity measure, I obtained the date when the first contribution to the proximity score occurred, by examining the timestamps for the overlapping events recorded in the working set pairs. I then compared that date with both the first day of concurrent work 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. Fig. 2 shows the probability density functions of proximity detection, CR detection and task duration in our main data set [1].

V. CONTRIBUTIONS

A socio-technical model constructed using developers' actions on artifacts as they occur and employing the proximity measure will provide an actionable and "live" view of CRs as they are established. This allows project governance decisions aimed at the resolution of CRs and prioritization of CRs that may improve productivity the most [9,18]. Based on the proximity measure, it is possible to devise tools that make developers aware of their coordination requirements as they happen.

VI. ONGOING AND FUTURE WORK

A plugin for the Jazz IDE [19], which implements the proximity algorithm and uses it to provide developers with visualizations of their coordination needs with co-workers, has been implemented. A live user study to assess how the tool and proximity measure support coordination in Jazz teams is under way.

The next step in my investigation of CRs is to determine if there are certain types of technical dependencies between software development tasks that do not require coordination. Currently, it is assumed that all work dependencies require coordination and generate CRs, but this is not necessarily true. If one could discriminate between inter-related tasks that require coordination and those that don't, the tool which implements the proximity algorithm could ignore the latter avoiding excessive coordination overhead and enabling developers to focus their attention on tasks where coordination and collaborative work is essential.

References

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

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

[3] 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.

[4] 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

[5] 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.

[6] 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.

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

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

[9] 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.

[10] Herbsleb, J.D. and Grinter, R.E. 1999. Splitting the organization and integrating the code: Conway's law revisited. Proc. ICSE 1999, 85-95.

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

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

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

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

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

[16] 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.

[17] Sosa, M.E., Eppinger, S.D., and Rowles, C.M. 2004. The misalignment of product architecture and organizational structure in complex product development. Management Science. 50, 12, 1674-1689.

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

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