Implicit Coordination in Software Development

Kelly Blincoe, Giuseppe Valetto
Department of Computer Science, Drexel University
Philadelphia, PA USA
kelly.blincoe@drexel.edu, valetto@cs.drexel.edu

Abstract— **This research proposal focuses on finding evidence of implicit coordination in software, analyzing the effects of that implicit coordination, and creating tools to further exploit the use of the inherently cheaper coordination means.**

Research Area/Subarea- Software Engineering. Social and Organization aspects of Software Engineering, Developers Coordination and Cooperation.

I. DESCRIPTION OF THE RESEARCH PROBLEM AND ITS IMPORTANCE IN THE FIELD

As software development organizations continuously grow in size, and become more global, coordination problems are amplified; consequently, the study of social and organizational dynamics in software engineering is increasingly gathering attention. For example, socio-technical congruence [3][4] provides the means to look jointly at the technical tasks of software development and the coordination requirements associated with these tasks. It offers a formal, quantitative means to analyze how much a project benefits (e.g. in terms of task performance) when its coordination requirements are met.

However, there is still much to discover in this area of software engineering. Many existing empirical studies on team coordination use explicit (and easily traceable) means of communication such as email, chat or meetings. We believe it is equally important to take into account other means of coordination. For example, forms of stigmergic communication [5] can occur in a software project in many ways. Developers can communicate implicitly via comments left behind in code, design specifications, change requests, commit logs, etc. [6]. This is arguably a cheap, efficient way to coordinate, for example, for developers that must overcome geographical barriers when interacting explicitly.

II. BACKGROUND AND RELATED WORK

Software is often developed in response to large complex problems, so the technical solution is usually not trivial. An attempt is made to design software in a way that it can be easily broken into small, manageable modules. Decomposing the software solutions in this manner allows developers to work in parallel on largely separate tasks [1]. However, since it is not possible to resolve all module interdependencies, developers must coordinate with each other to assure proper development boundaries, and to assure a working integration to the dependencies among their work. This coordination does not come cheaply; it can add to the project cost and delay schedules. Even worse, developers may not always be aware that they need to coordinate with each other. If coordination is insufficient there may be problems downstream in the development process, for instance when trying to integrate those modules in the final product.

To further complicate matters, the software industry is extremely competitive and organizations are forced to bid for new work aggressively. A project's originally proposed cost and schedule are often unrealistic. When a team begins to fall behind, often management tries to add additional members to the team to stay on schedule. While this may seem like a logical solution, it tends to complicate the situation, as observed by Brooks [2], because a larger team has increased coordination needs and must pay an overhead.

The issues above are manifestation of an essential complexity of software development, that is, its socio-technical nature [8]. That is particularly relevant in largely distributed projects, since remote interactions add significant overheads and delays [30][31][61]. Also, distance reduces the communication richness that people can experience: in particular, regarding informal communication, which has an important role in socio-technical systems [31][34][50]. According to [7], Software Engineering has traditionally approached these issues of coordination in three principal ways: trying to boost the individual productivity; or devising more efficient methods for the modular partitioning of work; or regulating the interactions among developers by strictly enforcing formal software processes. Lately, some researchers have turned to studying forms of implicit, coordination – such as stigmergic communication - because of their promise to substantially decrease the cost of the ineludible coordination overhead of software development activities [6][9][10]. This research proposal moves in that same general direction.

III. DESCRIPTION OF THE RESEARCH ISSUE/FOCUS

We propose a program of research to shed light on implicit means of coordination in software development by empirically studying software projects. The motivation is that a better understanding can lead to an enhanced and explicit support of implicit coordination. That, in turn, can yield advantages for projects that, like most Global Software Development efforts, face significant barriers or steep overheads with regard to explicit coordination means. An expected advantage is an increase in software productivity.

The first element of our research program aims at defining and cataloguing the means of implicit coordination that are commonly used in software development practice, and/or are emerging as an effect of innovations in technologies and tools used by developers, in particular in large-scale and globally distributed software production

environments. The second element of our program aims at detecting instances of implicit coordination in the course of project activities, then collecting and analyzing data about the level of implicit coordination occurring and its effects. Our main hypothesis is that projects with higher levels of implicit coordination will have higher performance in orchestrating and executing interdependent tasks successfully. We quantify higher performance here by a decrease in coordination overhead with no effect or an increase in software quality. If that hypothesis is correct, we would like to extend our work by looking at the relationships between explicit and implicit coordination, and investigate whether an increase in implicit coordination benefit performance even when it is not accompanied by additional explicit acts of coordination. Finally, the third element of our research program aims at creating tools to promote and support forms of implicit coordination across the software development process.

IV. PROPOSED METHODOLOGY

This kind of empirical research is best approached by a mix of methods, including ethnographic studies, data mining and analysis, and construction and deployment of tools that can help by validating research hypotheses:

- Observation of developers' behaviors and data collection through surveys and interviews will enable the first stage of the research program described above.
- The analysis of activity traces that are recorded in software artifacts and repositories can be a useful complement to those ethnographic observations. But its main role will be in enabling the collection of data about instances of implicit coordination. This kind of mining can occur *post mortem*, but also online, assuming the capability to instrument repositories and/or development tools in use within a project. The mined data will enable the verification of research hypotheses on the role of implicit coordination for development performance.
- Tools development has a twofold role. Tools can experimentally validate analysis results, for example, highlight what modes of implicit coordination are better used in various global development scenarios. Based on that, we can incrementally build a suite of tools for effective support of those scenarios.

V. EXPECTED CONTRIBUTIONS

If we could recognize when and how implicit coordination is used by software developers, and quantify its positive effect on teamwork and performance, then software development organizations could devise ways to systematically leverage these more efficient means of coordination and reap the corresponding benefits.

We foresee that, in the end, those benefits will be reified as new development tools, or new features in existing tools. An example could be a tool that collects comments in the code base and compares them to the comments in previous versions. Such a tool could verify whether developers are creating comments for the purpose of coordination. Comments that are changed could, for instance, be questions posed to other developers, or tags and flags meant to coordinate inter-dependent tasks, which are removed once addressed, or modified as this kind of stigmergic conversation continues. If it is found that developers employ this means for coordination, such a tool could be enhanced, e.g. by highlighting in some way this kind of comments for those developers who work on relevant tasks. This tool could also allow developers to create special types of comments that are explicitly created for this purpose, thus facilitating this channel of communication, and making it a first-class way of interaction with other developers. Many other tools like this one can possibly be devised, each of which could implement a different way to reduce the overhead costs associated with coordination in large-scale and global software engineering activities.

REFERENCES

[1] D. L. Parnas, On the Criteria to be Used in Decomposing Systems in Modules, Communications of the ACM, 15(2), pp. 1053-1058, 1972.

[2] F. P. Brooks, The Mythical Man Month, Anniversary Edition: Addison-Wesley Publishing Company, 1995.

[3] Marcelo Cataldo , Patrick A. Wagstrom , James D. Herbsleb , Kathleen M. Carley, Identification of coordination requirements: implications for the Design of collaboration and awareness tools, Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work, November 04-08, 2006, Banff, Alberta, Canada

[4] G. Valetto, M. Helander, K. Ehrlich, S. Chulani, M. Wegman, and C. Williams, Using Software Repositories to Investigate Socio-technical Congruence in Development Projects. In Proceedings of the ICSE International Workshop on Mining Software Repositories (MSR 2007), Minneapolis, MI, USA, May 2007.

[5] Elliott, M.A. Stigmergic Collaboration: The Evolution of Group Work. M/C Journal 9(2), 2006

[6] F. Bolici, J. Howison, and K. Crowston, Coordination without discussion? Socio-technical congruence and Stigmergy in Free and Open Source Software projects, in the 2nd ICSE International Workshop on Socio-Technical Congruence (STC 2009) Vancouver, BC, Canada, May 2009.

[7] Kraut, R.E. and Streeter, L.A. Coordination in software development. Communications of the ACM 38(3):69-81, March 1995.

[8] Emery, F.E., and E.L. Trist. Socio-technical Systems. In Management Sciences Models and Techniques , vol. 2. London, 1960.

[9] Heylighen, F. Why is open access development so successful? Stigmergic organization and the economics of information. In B. Lutterbeck, M. Baerwolff, and R. A. Gehring, (eds.),Open Source Jahrbuch 2007. Lehmanns Media, 2007.

[10] Robles, G., Merelo, J.J., and Gonzalez-Barahona, J.M.: Self-Organized Development in Libre Software Projects: A Model Based on the Stigmergy Concept. In: Proceedings of the 6th International Workshop on Software Process Simulation and Modeling, 2005.