Distributed cognition in software engineering research: Can it be made to work?

Jorge Aranda
University of Toronto
10 King's College Road
Toronto, Ontario, M5S 3G4, Canada
1-416-946-8878
jaranda@cs.toronto.edu

Steve Easterbrook
University of Toronto
40 St. George Street
Toronto, Ontario, M5S 2E4, Canada
1-416-978-3610
sme@cs.toronto.edu

ABSTRACT
Distributed cognition is a theoretical and methodological framework that considers social groups, their artifacts, and their contexts as a single cognitive entity working towards the solution of a shared problem. In this paper we briefly describe the framework and consider its strengths and weaknesses as a theoretical foundation for software engineering research. We propose a series of techniques to address the methodological problems that the application of the framework entails in our research field. Finally, we present an ongoing exploratory case study that aims to evaluate the adaptability of the framework and of the techniques we propose here.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *programming teams, software process models.*

General Terms
Documentation, Experimentation, Human Factors, Theory.

Keywords
Distributed Cognition, Social Networks Analysis, Artifact Analysis, Empirical Software Engineering.

1. INTRODUCTION
Research of software engineers at work –of their team structures, interactions, and dynamics– has been largely performed as a butterfly collection exercise: We have many interesting bits of results, but we do not have a theoretical framework that links the separate phenomena we observe, unifies our perspectives of the domain, and allows us to generate testable predictions of software projects and software teams. As a consequence, our findings are not exploited to their full potential; and our research effort is often spent exploring unviable or shallow hypotheses [6].

For illustration purposes, consider the extensive literature on design and code inspections [7]. Although there have been dozens of studies testing the phenomenon, they provide little insight as to why it occurs, how can its beneficial effect be amplified, and what could possibly be the consequence of performing inspections in ways that have not been empirically tested. The reason, we claim, is that until recently inspection studies were not designed over a theoretical foundation that predicted their effects and addressed these issues. If inspection researchers had a theory to guide their work, they could have spent their efforts validating it and probing its predictive power, yielding even stronger findings for our domain.

The inspections literature is the norm, not the exception, when it comes to theory building and theory validation in the software engineering realm. To address this problem, we are evaluating the capabilities of a theoretical framework (distributed cognition) and its applicability to software engineering in an exploratory case study. In this paper we present the gist of the distributed cognition theory, its strengths and weaknesses with regards to software engineering research demands, and the adaptations we feel are necessary for such a framework to be convenient for our research. We also briefly describe the case study we are conducting and the roadmap we intend to follow in the near future.

2. DISTRIBUTED COGNITION
Distributed cognition is an interdisciplinary theoretical framework designed to study cognition as it occurs in socially situated contexts. Its unit of analysis is the functional system of people and artifacts in charge of executing a cognitive task. That is, for a distributed cognition researcher, the functional system is a single cognitive entity, and although no element within this entity may know how to solve the cognitive task, the full range of interactions and transformations of information within the group produce a workable solution to the cognitive problem at hand [11]. Perhaps the classic example of distributed cognition research is Hutchins' study of sea navigation [4], where each person in the navigation team of a ship performs a set of simple tasks based on their role and the information available to them, and although no person in the team has a full knowledge of the situation, the end result is a calculation of the ship's position in the world.

Although the distributed cognition framework is too extensive to be summarized here, there are several properties about it worth mentioning. First, it centers on the study of situated cognitive activities, as opposed to artificial laboratory settings. According to

the theory, cognitive performance should not be analyzed in constrained settings, since much of people's real cognitive work is done by the interaction among them and with their context.

Second, artifacts are viewed as embodied knowledge –they store rules and processes that simplify the cognitive tasks of their users. Therefore, analyzing the artifacts people use is an essential aspect of the framework.

Third, identifying the paths that chunks of information follow to reach the persons that need them is a key consideration of distributed cognition work. Team members that work on a cognitive problem start up with different bits of knowledge, and an important step towards solving the problem is to share and transform them, through mediated or direct communication, until they reach the person who needs them.

Finally, the framework studies cognitive work on two different levels: In the short term, it focuses on the actual resolutions of cognitive problems; while in the long term, it analyzes the learning and structuring activities that take place in teams.

Since its original formulation, the framework has been used to examine a wide variety of groups and contexts, including navigation [4], aviation [5], hotline centres, rescue teams, and, in one occasion, software developers performing maintenance tasks [1]. Unfortunately, so far there have only been a few teams applying the framework and producing this research –most notably Hutchins' own research group at San Diego.

The distributed cognition framework is still far from being generally accepted by any research community. In the CSCW literature, a response by Bonnie Nardi to a paper on theories for CSCW [3] critiques several theoretical and practical problems of the framework [10]. She points out how its insistence on ethnographic methods, and in particular of ethnomethodology, causes an "anemic theoretical development", which, she warns, leads to "a withering of community in any field of study." She also notes that distributed cognition, as proposed by Hutchins and in parallel to ethnomethodology, is suspicious of conceptual elaboration, undermining communication and comprehension efforts in the research community.

3. DISTRIBUTED COGNITION IN SOFTWARE ENGINEERING

3.1 Applicability of the theory: Benefits and drawbacks

The idea of conceptualizing software development as a socially distributed, artifact-intensive cognitive activity is compelling, and we believe the software engineering field could reap important benefits by adopting this view. Here are some of the advantages that result from appropriating this theoretical foundation:

- A systemic view of software teams, which includes the social aspects of team collaboration and the study of the interactions between humans and their artifacts. All software development practices, documents, and tools, can be re-interpreted and explored within this view.
- An abstraction of all interactions and uses of artifacts as *transformations of representational states across representational media* [4], which allows for evaluating the effectiveness of alternative transformations by interpreting software development techniques (such as code reviews, pair programming, and prototyping) as transformations and representations of information with particular coordination- and communication-related strengths and weaknesses.
- An emphasis on analyzing artifacts both as embodied knowledge and as communication media, leading to insights about new and modified proposals for tools and languages to capture and transfer that knowledge.
- A consideration of individual and organizational learning, role specialization dynamics, and the context in which these phenomena take place, which may prove to be a fruitful perspective for software project management research.

Both in general, as a paradigm of the software development field, and in particular, as a collection of techniques for improving the context and tools in which cognitive-intensive activities take place, distributed cognition seems to be a useful perspective to adopt for software engineering research. However, if it is to become a theoretical foundation for this research, it will need to undergo significant methodological alterations to achieve practicality.

We think software engineering research cannot be built over an ethnomethodological foundation. Ethnomethodological studies are necessarily constrained to the analysis of particular, detailed phenomena, and the amount and variability of such phenomena in software projects is overwhelming, even for small-scale projects. It boggles the mind to consider how a comprehensive ethnomethodological study, of the kind performed in the distributed cognition literature, could be carried out in a large-scale, geographically distributed, multi-year development project.

To turn the framework into a feasible alternative for this type of research, we need methods that abstract away some of the details of day-to-day phenomena and focus on detecting the essential patterns of communication, team structure, and artifact use in software projects. Before proposing any methods, however, we must address the question of whether such departures from ethnomethodology are compatible with the core ideas of distributed cognition or, alternatively, ethnomethodological detail is an essential component of the framework.

It seems to us that ethnomethodology is, though valuable, accidental to the theory; a result of the background of the original distributed cognition researchers. Just as it might be desirable for cognitive scientists (but impractical under our technological and practical circumstances) to examine every synapse in the brain, analyzing every utterance of a problem-solving group is not essential to the conceptualization of such group as a distributed cognitive entity.

What, then, is essential? To get a basic picture of a distributed cognitive system, at least the following elements need to be analyzed:

- Group structure and patterns of group interaction
- Artifacts (tools, documents), and patterns of artifact use
- Nature and frequency of tasks

- Development of shared understanding, breakdowns and recoveries

There are techniques, both from distributed cognition and from other disciplines, to study these types of information. In the next subsection we propose some of the most promising ones.

3.2 Social Network Analysis (SNA)

Sociologists have developed a collection of methods to analyze the structural and dynamic qualities of social groups [13]. We do not have the space to describe them in detail, but we would like to mention a short list of them. To start, social network graphs and simple SNA measurements such as *centrality* and *density* provide an initial overview of the structure of a group. More elaborate techniques, such as *blockmodelling* (for clustering nodes based on their similarities in several networks) and *positional analysis* (for simplifying the information in network data sets), among others, complete the picture of group structures. Finally, other SNA-inspired concepts, such as knowledge transfer and social capital, add fruitful perspectives to the study of group interactions.

Some kinds of software projects are particularly amenable to SNA methods –those for which communication takes place almost exclusively in electronic form, such as most open source projects [9]. In these cases, a full record of interactions is available to the researcher, and one can track the proposal of new ideas, the types and frequency of contributions, and the transfer of information among project members. For other projects, particularly those in which participants are collocated, many exchanges of information and much knowledge of the social structure of the group is not recorded electronically or in the project's documentation, and must be extracted directly from participants.

However, an important advantage of SNA methods for our purposes is that the data they require are relatively easy to collect. Conducting case studies to understand the full structure of software development teams becomes feasible, and surveys of wide ranges of software houses are also possible.

On the other hand, SNA methods were designed for sociological goals, and they are often concerned with topics that are not of immediate relevance to software engineering, such as power relations, social support, and the job market. To our knowledge, no study has yet analyzed in detail the implications of applying the methods of SNA to the software engineering field.

3.3 Artifact analysis

Some of the most satisfying results from distributed and external cognition studies are their analyses of artifacts people use to perform their tasks. Through these analyses we discover how cognitive activities are simplified by representing information and rules "in the world", rather than in people's heads [15], and by re-representing complex information in ways that simplify its understanding [12].

For software development projects, artifact analyses may provide insight into the efficacy and dynamics of document and tool use. For documents (a category in which we include, for instance, specifications, models, and emails), the researcher may find what information flows among people, how expressive, efficient, and useful are the representations, how quickly do they become obsolete or out of sync with the world, and what are the skills necessary to create them, modify them, and read them.

The thorough study of all documents used in a project is not practical. But collecting data on the frequency with which different types of documents are used and their relevance for each group member provides us with useful patterns of interactions and of team dynamics. It will also point to particularly relevant documents, which may be studied with the more careful detail that traditional distributed cognition literature displays.

For tools, of which every programming language, IDE, project website, and debugger are examples, the researcher may uncover cognitive benefits provided by new and existing proposals based on the computational effort they demand from their users.

Tool analyses are detailed and time-consuming. However, once performed, their findings are applicable for projects that use the same tools under similar settings, paying off the investment considerably.

3.4 Other approaches

We are evaluating the utility and practicality of other approaches to support distributed cognition in software engineering; approaches that in principle can be effective complements to SNA and artifact analysis, but whose empirical validity is still not clear.

One such alternative is *conceptual sketching* [2], which may provide rich details about the networks, perceptions, and mental models of participants of a software team. Conceptual sketching, however, may also be prone to misinterpretations and vague results, which are, of course, undesirable characteristics in software engineering research.

4. CASE STUDY

To test the viability of the distributed cognition framework and the methodological adaptations we propose, we are conducting a pilot case study on the release team of a software division at IBM. This work feeds upon other studies of developers, such as that of LaToza et al. [8], and other attempts to conciliate software engineering and distributed cognition [14].

The release team is a high-impact, high-interaction volume group within the division. It oversees product development and serves as a bridge between "technical" and "business" people. This bridge role requires from them, in addition to advanced project management skills, a familiarity with at least two different professional cultures, vocabularies, and goals. They are focal enablers of shared understanding in the division, in the sense that they are the main point of contact for developers to learn project requirements, and for managers to learn their projects' status.

For these reasons, the people at the release team have experienced the need to create roles, team dynamics, and processes that help them handle their responsibilities and coordinate the efforts of the full division towards shipping their releases. We think the analysis of these roles, dynamics, and processes, with a distributed cognition lens, should be particularly insightful.

We designed our case study to explore these phenomena. We decided to interview every member of the team with a structured questionnaire that probes the techniques we described above. Our interview has four main sections. First, we ask participants to draw conceptual sketches of their team, of their interactions with other teams, and of their division within and outside the company. Second, we collect social network data, focusing on several types

of personal networks (information consumers and producers, collaborators, mentors, and informal networks). Third, we ask participants to describe the main activities they perform according to their role, and to list the artifacts (documents and tools) that they use to perform each of these activities. Finally, we ask them open-ended questions about the goals of their role and their team, success criteria, success factors, and an overall description of their position in the company.

Each section of the interview will first be analyzed separately, and their findings will later on be put together to detect patterns among them. We designed the questionnaire in a way that allows us to evaluate both the team itself and the methods we chose to use, so we can refine them for future larger-scale case studies.

We are, at the moment of writing, in the data collection phase of our case study. We have collected the data of nine participants, with five more to go. We will proceed to analyze their conceptual sketches and their social networks data separately, and to identify the most relevant tools and documents they use in order to perform an artifact analysis on them.

After refining our techniques with findings of this case study, we plan to conduct at least two other studies in the same organization. The first is an extension of our current study – including data from the technical and business groups that interact with the release team we are analyzing. The second is a replication of our initial study, for a different release team, in an effort to detect the patterns that arise from two divisions with different cultures within the same corporation.

As an end result of these empirical studies we expect to obtain two types of benefits: For the organization, we should be able to produce recommendations for tool, document, and process improvements. For our research team, we will have data regarding the viability of the methodological approaches we describe in this paper, and the adaptations we find necessary for their successful implementation by our research community.

5. CONCLUSIONS

Distributed cognition is a fruitful foundation to support research of software engineers at work, but if it is to be used for this purpose, we need to overcome its methodological constraints with alternatives such as the ones discussed in this paper. We believe that, by rejecting the notion that we can (or should) capture and analyze every detail of the interactions of developers, software engineering research can benefit greatly from the perspectives the theory provides while allowing studies of the social side of software development to remain feasible.

We think that the methods and techniques we described above can support empirical studies of this kind by substituting the ethnomethodological studies of traditional distributed cognition with workable solutions that still enable us to make key findings. However, we do not have any data to back up these claims yet. Our pilot case study, and possible subsequent studies, will allow us to make an evaluation of which of the techniques we propose are off the mark, which need some adaptation, and which work well for our field.

6. ACKNOWLEDGMENTS

We would like to thank and acknowledge the financial support from Bell University Labs at U of T. We are also very grateful to IBM for supporting and cooperating with our study.

7. REFERENCES

[1] Flor, N.V., and Hutchins, E. Analyzing Distributed Cognition in Software Teams: A Case Study of Collaborative Programming During Adaptive Software Maintenance. *In Koenemann-Belliveau, J., Moher, T., and Robertson, S. (Eds), Empirical Studies of Programmers: 4th Workshop*, 1992.

[2] Goel, V. *Sketches of thought.* Cambridge, MA: MIT Press. 1995.

[3] Halverson, C.A. Activity Theory and Distributed Cognition: Or what does CSCW need to DO with theories? *Computer Supported Cooperative Work, 11*, 243-267, 2002.

[4] Hutchins, E. *Cognition in the Wild*. MIT Press, Cambridge, MA, 1995.

[5] Hutchins, E. How a Cockpit Remembers Its Speeds. *Cognitive Science, 19*, 265-288, 1995.

[6] Jørgensen, M., and Sjøberg, D. Generalization and Theory-Building in Software Engineering Research. *Proceedings of the Workshop on Empirical Assessment in Software Engineering (EASE'04)*, 2004.

[7] Laitenberger, O., and DeBaud, J.M. An encompassing life cycle centric survey of software inspection. *Journal of Systems and Software, 50, 1,* 2000.

[8] LaToza, T., Venolia, G., and DeLine, R. Maintaining Mental Models: A Study of Developer Work Habits. *Proceedings of the International Conference on Software Engineering (ICSE'06)*. (Shanghai, China, May 20-28, 2006).

[9] Madey, G., Freeh, V., and Tynan, R. The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory. *8th Americas Conference on Information Systems*, 2002.

[10] Nardi, B.A. Coda and Response to Christine Halverson. *Computer Supported Cooperative Work, 11*, 269-275, 2002.

[11] Rogers, Y., and Ellis, J. Distributed Cognition: an alternative framework for analysing and explaining collaborative working. *J. of Information Technology, 9, 2,* 119-128, 1994.

[12] Scaife, M., and Rogers, Y. External cognition: how do graphical representations work? *International Journal of Human-Computer Studies, 45,* 185-213, 1996.

[13] Wasserman, S., and Faust, K. *Social Network Analysis: Methods and Applications.* Cambridge U. Press, 1994.

[14] Ye, Y. Supporting Software Development as Knowledge-Intensive and Collaborative Activity. *Proceedings of the 2nd Intl. Workshop on Interdisciplinary Software Engineering Research (WISER'06).* (Shanghai, China, May 20-28, 2006).

[15] Zhang, J., and Norman, D. Representations in Distributed Cognitive Tasks. *Cognitive Science, 18,* 87-122. 1994.