Discovering the Shared Understanding Dynamics of Large Software Teams

Jorge Aranda
University of Toronto

Ramzan Khuwaja
IBM Toronto Lab

Steve Easterbrook
University of Toronto

Abstract

Reaching project goals demands from team members the creation and communication of detailed and vastly heterogeneous project information. Although no team member needs to know every piece of project information, each of them depends extensively on knowledge generated by other parties. Their aggregated information-seeking and information-sharing activities form a web of interactions that develops the team's *shared understanding* of their project.

Current approaches to study this phenomenon are unsatisfactory, as they tend to overlook its inherent complexity. To address this issue, we present a proposal to analyze shared understanding dynamics that draws from cognitive and organizational theories, as well as from Kruchten's 4+1 views of software architecture.

1 Shared Understanding

Reaching project goals demands the creation and communication of detailed project information among team members. The nature of this project information is vastly heterogeneous: it may be as overarching as requirements specifications or as seemingly trivial as the location of a file in a repository. Usually, no team member needs to know every piece of information, and in large projects it becomes impossible to have total knowledge. However, every team member depends on knowledge generated by people other than themselves, and their aggregated information-seeking and information-sharing activities form a web of interaction with one main purpose: improving the team's *shared understanding* [4] of their project.

Reaching this shared understanding is considerably difficult, even for small teams. It involves the effective use of social, organizational, and cognitive strategies. Shared understanding cannot be developed systematically, and it can never be *guaranteed* because we cannot ensure that two parties have a shared understanding. We can only point to interactions in which the lack of understanding became apparent through breakdowns and conflicts.

Developing shared understanding becomes a monumental task for large-scale software development, where software teams are geographically separated, their members number in the hundreds or thousands, and their skills and vocabularies are extremely specialized and widely divergent. But the difficulty of the task does not make it any less vital. Without reaching enough shared understanding, a software team will not be working consistently towards the same goals, and it will be surprised by obstacles that were not flagged and communicated in advance. In contrast, the development of shared understanding will focalize the efforts of the full team, improve the flow of information within and from outside its boundaries, and improve the organization's opportunities in the market. Reaching shared understanding, then, is a necessary, if not a sufficient, requirement for the success of a software development team.

The researcher of shared understanding in software teams encounters three problems from the start:

Completeness: It is realistically impossible to record and analyze all of the relevant data of teams of even modest sizes.

Observability: It is impossible to *confirm* that two people share the same understanding of a

situation. It is only in moments of breakdown that a lack of understanding becomes evident.

Breadth of focus: Shared understanding can be studied with a variety of strategies, and each will only display one angle of the problem. An exclusive focus, for instance, on processes and methodologies will ignore the informal interactions (such as water cooler conversations) that play a large role in strengthening the understanding in a team; a focus on the latter will equally suffer from ignoring the processes that, unknown to the researcher, every team member has internalized as part of their understanding.

As a response to these challenges, we have developed a multiple-view approach to study the development of shared understanding in large software teams. Details of our proposal are summarized in the following section.

2 Organizational Views

In order to study the development of shared understanding, we need to ground our analysis on a framework that incorporates several key *views* that offer different insights on the software team.

This is not the first time that studies in our field have needed to combine multiple aspects of analysis. The success of a similar approach in software architecture should be of use for the study of shared understanding. We refer to Kruchten's "4+1 view model" of software architecture [2]. Kruchten identifies the problem caused by the narrowness of several approaches to model architectures: *"Sometimes the architecture of the software suffers scars (...) from an over-emphasis on one aspect of software development."* In response, Kruchten proposes *"to organize the description of software architecture using several concurrent* views, *each one addressing one specific set of concerns."*

The concurrent views for software architecture proposed by Kruchten are summarized in Figure 1.

The complexity of the phenomenon of shared understanding, and the qualities of Kruchten's model, suggest that an approach similar to his own should be fruitful for our domain. A diversity of views is needed when analyzing something as complex as large-scale software development. These views need to cover the technical, social, cognitive, and organizational factors that impact software development, and their combined application should convey a holistic picture of the software development team under study.

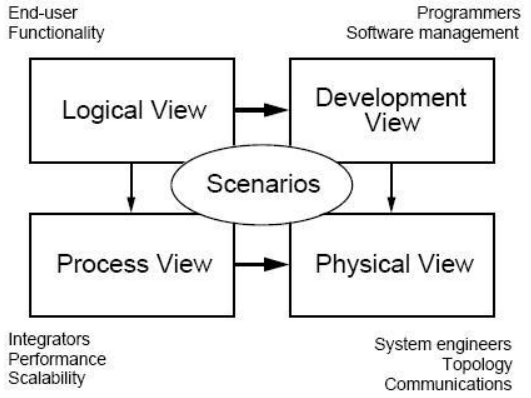

Figure 1 - Kruchten's 4+1 view model of software architecture

As part of our studies of shared understanding, we have prepared a preliminary version of this multiple-views model of organizations. Our proposal is the result of a survey of the literature in the areas we mention, as well as of a pilot study of a project management team, which we cannot report here because of space limitations. The model has not yet been properly evaluated, and we expect it will suffer significant changes as we progress with our research. However, we consider it helpful to describe it at this stage in order to begin a dialogue with the community regarding the convenience of each of the views we describe, and the best approaches to analyze them.

All of these views have been proposed previously, in separate, for the study and understanding of organizations. To our knowledge, however, they have not been proposed under a unified framework before. Our proposal is composed of the following views:

Structural view: This view captures the hierarchy of the software development organization, the role of each member, and the structure of its work groups. This view is trivial for small teams, but large groups have complex hierarchical structures that affect the development of software deeply. When individuals need to satisfy an *n*-dimensional matrix management structure, we need to understand said structure if we want to propose sensible refinements to the organization.

Process view: This view describes the business processes and methodologies followed by the organization. It is, perhaps, the most commonly

used perspective to describe an organization in the IT industry, and it is also the one that has been better developed –languages such as BPML can be used extensively to document this type of information.

We propose to focus on processes as *executed* instead of as *specified* because this focus will make apparent the shortcuts and tacit activities that organizations perform but do not usually document.

Dependency view: This view records the chains of dependencies between the product under development and other products from within or outside the organization.

Software products in large companies have many dependencies to other projects. Their successful completion depends on the completion of products built by other teams, which in turn depend on others. The chain of dependencies becomes quickly complicated and volatile.

Although some companies have successfully found ways to document and keep track of the dependencies of their projects through dependency management applications, many companies still struggle with this challenge. Studying the development of shared understanding for these companies requires the analysis of product dependencies, of the mechanisms through which these are discovered and negotiated, and of the ways in which they are managed as part of product development.

Social view: This view focuses on the formal and informal interactions between stakeholders of a software project, and on the social structures in which they are arranged.

Through the use of Social Network Analysis [5], we can analyze the structural and dynamic qualities of software teams, and explore the ways in which interactions take place in practice (as opposed to the ways in which they are prescribed by business processes).

The study of social networks is of particular relevance to large-scale software development teams because many of their members serve the primary goal of facilitating the flow of information among social clusters. These individuals bridge the gaps between groups with different vocabularies, skill sets, and expectations. As a research community, we know very little about how these bridges carry out their tasks and ensure that product development runs smoothly.

Competencies view: This view provides an inventory of the skills, background knowledge, and experience of the members of the software team. It is relevant because large teams develop highly specialized roles and competencies, and these usually imply distinct terminologies and subcultures. An awareness of these personal differences will be of importance when restructuring teams, providing opportunities for learning, and analyzing the feasibility of the implementation of best practices and team dynamics.

Artifact view: This view allows us to investigate the qualities, affordances, capabilities, and limitations of the artifacts used in everyday software development [3]. These artifacts (documents such as requirements specifications, or tools such as IDEs) are often focal points of communication and cognition, and hence their characteristics have a significant impact in software development activities.

The thorough study of all documents and tools used in a software project is not practical. However, collecting data on the frequency with which different types of documents are used and their relevance for each group member may provide us with useful patterns of interaction and team dynamics. It will also point to particularly relevant documents, which may be studied with a more careful detail.

Intentional view: Tied to the social actors and to their positions in the organization's structure, this view makes explicit the key goals (and means to achieve them) of a socio-technical system's agents. This perspective allows us to address questions such as these: What are the goals of each agent? Are they currently satisfied or denied? What are the consequences of the interactions of the actors? And, perhaps more importantly, which structural arrangement suits best the satisfaction of as many goals as possible?

The study of the intentionality of social agents has advanced considerably. Some current proposals, such as the *i** framework [6], provide analysts with the tools to apply the intentional view to socio-technical contexts. At the same time, large-scale uses of these proposals have so far resulted in cumbersome intentionality models that do not facilitate analysis. Considering the overwhelming number of agents and goals that are involved in large-scale software development, refinements to these intentionality proposals appear necessary if they are to be used for the study of shared understanding in these contexts.

Physical view: This view notes the geographical location of the team members, and the layout of their work environment.

This is perhaps the most overlooked perspective in our field, even though the software development issues that arise because of lack of physical proximity are considerable [1]. There are indications that these issues are, indeed, determinant factors of success or failure of software development projects.

Scenario-based view: The eight views proposed so far are valuable by themselves, but they are likely to interact in ways that our separate perspectives cannot convey. Therefore, we propose an additional view, the equivalent of Kruchten's "+1": a scenario-based view that ties the rest together, providing a unified perspective of software development.

Because of its unifying character, this is perhaps the most important of all the views described here. It explains how an event is handled from its initial trigger to its final consequences. It represents, in a way, "executing" the socio-technical system, and tracing it along all the other views of our proposal.

By necessity, the scenario-based view will be incomplete. It is impractical to capture the details of the execution of every event through the organization. We should be concerned with the normal and extraordinary execution of a few key scenarios that illustrate an aspect of the business's operation, not with the totality of scenarios and their consequences. But this incompleteness is not a deficiency exclusive to our proposal: any other approach to the study of large-scale software development will need to ignore some data as well.

3 Current and Future Work

We are presently researching shared understanding at the scale of a large software division following the framework described in this paper.

We expect the result of this exercise to be twofold. First, we should generate a partial map of shared understanding in the software division. The map should be useful in the detection of patterns of shared understanding dynamics, leading to the identification of best practices, tool and document improvements, and mechanisms for the prevention of breakdowns in software teams.

The second outcome of this exercise should be the refinement of this model of organizational analysis, which will be potentially useful not only for researchers of shared understanding in software teams, but for many other software project management studies.

Acknowledgements

This research was supported by IBM's Centre for Advanced Sciences (Toronto).

About the Authors

Jorge Aranda is a Ph.D. student in Computer Science at the University of Toronto. His research interests are the social and psychological aspects of software development. He can be reached at jaranda@cs.toronto.edu.

Ramzan Khuwaja is a Project Manager currently assigned to IBM's Software Group, Rational Tools, project management office. Ramzan has a Ph.D. in Computer Science, and 13 years of working experience in the IT industry. He can be reached at ramzank@ca.ibm.com.

Steve Easterbrook is a Computer Science professor at the University of Toronto. Contact him at sme@cs.toronto.edu.

References

[1] T. DeMarco and T. Lister. *Peopleware: Productive Projects and Teams.* Dorset House Publishing Co., 2nd edition, 1999.

[2] P. Kruchten. "Architectural Blueprints—The "4+1" View Model of Software Architecture" *IEEE Software* 12(6), Nov. 1995.

[3] M. Scaife and Y. Rogers. "External cognition: How do graphical representations work?" *Intl. Journal of Human-Computer Studies,* 45, 1996.

[4] L. Suchman. *Plans and Situated Actions: The problem of human-machine interaction.* Cambridge University Press, 1987.

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

[6] E. Yu. "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering" *Procs. of the 3rd IEEE International Conference on Requirements Engineering (RE'97),* 1997.