Towards Awareness in the Large

Anita Sarma and André van der Hoek
Department of Informatics
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697-3440, USA
{asarma, andre}@ics.uci.edu

Abstract

Management of shared artifacts is critical to ensure the correct integration and behavior of code created by multiple teams working in concert. Awareness of inter-team development activities and their effects on shared artifacts provides developers the opportunity to detect potential integration problems earlier and take proactive steps to avoid these conflicts. However, current awareness tools do not provide such kinds of awareness making them unsuitable for global software development. In this paper, we discuss their drawbacks, present three strategies to make them suitable for global settings, and illustrate these strategies through a new view for Palantír that better addresses awareness in the large.

1. Introduction

Awareness is characterized as "an understanding of the activities of others, which provides a context for one's own activities" [5]. The concept of awareness has since long been socially employed at the workplace to facilitate coordination. For example, employees generally use informal hallway chats and coffee-hour discussions to keep abreast of the latest "happenings" in the organization. Another example is the use of instant messaging in the workplace, which enables distributed developers to participate in informal conversations with their colleagues, be aware of how active they are, and when they can be approached [7, 11].

Lately, the concept of awareness has received significant attention in coordination of (distributed) software development. Numerous coordination technologies have evolved that enhance the coordination capabilities of Configuration Management (CM) systems by promoting awareness of development activities of the team. The hypothesis behind these coordination technologies is that awareness of parallel activities allows developers to place their work in the context of others' changes, which enables them to identify potential problems earlier and gives them the opportunity to self-coordinate their actions to avoid these problems.

While some of these tools (e.g., CVS-Watch [2], COOP/Orm [10], BSCW [1]) provide awareness of activities at the repository level, others take a step further and provide real-time information of ongoing changes in remote workspaces (e.g., Palantír [14], JAZZ [3], NightWatch [12]). These coordination technologies typically provide information regarding changes to artifacts along with some meta-information, such as which artifact is being changed by which developer, whether an artifact is being changed in parallel, and whether the changes would cause artifacts in the local workspace to be out-of-sync. This information is generally displayed either through separate contextualized visualizations or through embedded awareness widgets in the development environment.

Thus far, awareness tools have been typically designed for the individual developer coordinating with her immediate team. These tools generally portray information of changes to artifacts that are either present in the developer's local workspace or artifacts in which she has specifically registered interest. While these tools help in understanding individual changes to specific artifacts they fail to provide a global understanding of interactions among teams, such as which artifacts are shared among teams, which teams are tightly coupled, what is the effect of a change to a shared artifact on other teams. This drawback makes current awareness-based coordination tools ill-suited to software development that involves multi-team development.

A significant hurdle in adopting existing awareness-based coordination technologies to, what we term,

awareness in the large is the scale of operations in global software development. Current tools are not designed to handle the large amounts of information generated in operations of multiple teams and, therefore, cannot adequately manage its smooth transmission or, more importantly, the cognitive overload created by it. In addition to scaling problems, awareness technologies in global software engineering have to contend with global development specific issues such as inter-organizational cultural differences, privacy concerns, and time-zone differences.

We draw upon our experience in building and using Palantír (a workspace awareness tool for distributed software development) and literature review of problems manifesting global software engineering to present strategies for the collection, analysis, and visualization of information regarding inter-team development activities to promote awareness in the large.

1. *Information Abstraction*: Information of development activities needs to be abstracted to present a high-level view of team interactions.
2. *Impact Analysis*: The impact of changes on shared artifacts should be tracked to detect potential conflicts arising due to changes made by a team that adversely affects other teams.
3. *Specialized Visualizations*: Visualizations that present a comprehensive view of team dynamics such as team locations and interdependencies among teams.

We use the aforementioned strategies to design the World View, a new view that we plan to build into Palantír to enable it to handle coordination of multi-team development.

The rest of the paper is organized as follows. In Section 2, we discuss the need for awareness in global software development and the drawbacks of current awareness-based coordination technologies. Section 3 describes the strategies for awareness in the large followed by illustration of these strategies through Palantír in Section 4. We present our conclusions in Section 5.

2. Motivation

In global software development, the management of shared artifacts (artifacts common to multiple teams) is critical to ensure the correct integration and behavior of code written by different teams. Organizations strive to minimize the interdependencies among teams by: (1) explicitly structuring the software to ascertain loose coupling among teams and (2) designing the interfaces through which teams interact well in advance [8, 13].

In real life, however, it is seldom possible to avoid dependencies among teams or to ensure the immutability of interfaces. Empirical studies conducted at several software development companies with global operations have observed that shared artifacts are frequently changed and communication of these changes across teams is not well-supported. For example, De Souza et al. found that developers across teams depended on the immutability of Application Programming Interfaces to ensure that their work integrated with each other [4]. However, these interfaces were frequently modified and the information of these changes was not always communicated across teams. In another study [6], Grinter observed that developers at a multinational company identified other developers who were dependent on their piece of code by broadcasting email messages and by collecting responses to those messages. The weakness of this approach, apart from relying on an ad hoc communication mode, was that dependency relationships were maintained by individuals who owned each component and when that individual left the team the connection was lost. Grinter observed that organizations frequently created separate repositories and organizational units to specifically manage shared code, such that modifications to shared code was governed by strict access protocols, sequential development, and formal modification requests approved by the special organizational unit [6].

Awareness-based coordination tools have gained significant attention in the recent past. However, the technical solutions that exist today are primarily geared towards small teams [6, 13]. These technologies typically promote awareness by presenting fine-grained information of development activities and the status of individual artifacts in specific projects. For example, such tools usually notify developers of when an artifact has changed, whether the artifact has been concurrently modified, or whether the changes cause artifacts in the local workspace to be out-of-sync. While such information is invaluable for placing one's work in the context of changes taking place within a team, it is significantly less useful for inter-team interactions. In a well designed project, members of a team rarely need detailed information of changes to artifacts within another team, unless those artifacts are shared; such detailed information, in fact, leads to the cognitive overload of users.

Moreover, these technologies are designed such that only information of activities regarding artifacts that are either present in the developer's local workspace or those in which the developer has specifically registered interest is tracked. In global settings, teams rarely check-out code-based artifacts that belong to projects of other teams; instead they work with binaries of the

components. Additionally, a significant problem in coordination of multiple teams is the identification of shared artifacts and teams that depend on those artifacts [4, 9]. In global settings, it is unrealistic to assume that developers would be able to identify a comprehensive list of shared artifacts on which they depend in order to register interest.

3. Strategies for Awareness in the Large

In global software development, the software under construction is structured so as to minimize the interdependencies among teams and programming interfaces are defined well in advance for cases where code produced by different teams need to interface. Generally, multi-team projects design "shallow" interfaces – test interfaces with only method signatures and no underlying implementation – against which teams test their code while the "real" interfaces are under implementation [4]. In such settings, teams are not aware of changes to the "real" interfaces unless they are directly notified or it is time for the final integration of the software system.

Since the integration of code produced by a team greatly depends on the immutability of the interfaces and shared artifacts, it is critical that teams be aware of modifications to these artifacts. Therefore, one of the primary goals of tools for awareness in the large is to identify shared artifacts that transcend team boundaries, track changes to these artifacts, and notify teams that are affected by these changes.

In the following sections, we discuss strategies for *collecting*, *analyzing*, and *visualizing* information of development activities, to promote awareness in global settings, which we term "awareness in the large".

3.1. Information Collection

Strategy 1: Abstract information to present a high-level view of development activities.

Current awareness tools typically provide detailed information of development activities. Traditional coordination tools provide information at the level of mouse-clicks and keyboard strokes; whereas CM-based awareness tools provide information at the individual file or method level. The level of detail required depends largely on the type of collaborative effort; fine-grained information of every user action is beneficial in synchronous editing, while individual artifact-level information is helpful in tracking modifications to artifacts that are present in the local workspace. In global software development such detailed information is not suitable since it leads to excessive amounts of information that cognitively overloads the user.

Moreover, in such settings, teams rarely check-out code-based artifacts of other teams and generally work with the binaries of components on which they depend. Since, current awareness tools typically provide information of changes pertaining to artifacts present in the local workspace they may fail to capture information of changes to other artifacts that may have an effect.

Awareness in the large, therefore, requires information at a higher level of abstraction to identify those development activities that affect other teams. A principal step towards this goal is the identification of shared artifacts. A note to consider is that such identification should be based on the underlying code constructs and not just the design documentation to ensure that all shared artifacts are accurately identified. Once these artifacts are identified, changes to these artifacts should be tracked and information regarding these changes disseminated.

3.2. Information Analysis

Strategy 2: Analyze changes to determine the impact of a change on other teams.

Existing awareness-based coordination tools generally provide information of which artifact is being changed by which developer. This information is effective in informing developers of when an artifact has been modified, whether an artifact is being concurrently modified, and which developer is responsible for which change. Although this information is helpful in allowing a developer to place their work in the context of other changes in the project, it cannot help the developer identify indirect conflicts – conflicts that arise due to a change in one artifact affecting another. For instance, if an interface or a library item is modified that change affects all those other artifacts that use the interface or depend on the library item. Current awareness tools are unable to detect such indirect conflicts.

Since multi-team development hinges on the immutability of shared artifacts (e.g., interfaces, libraries, modules for reuse) it is imperative to calculate the impact of a change to a shared artifact and to notify affected team(s). Additionally, metrics to denote the significance of an indirect conflict (e.g., number of artifacts affected, number of teams affected, size of the change) should be calculated. Advance warning of such conflicts serves two purposes. First, it allows the developer who is making the change to understand its significance and provides him the opportunity to communicate information of the change to teams that will be affected. Second, it serves as a warning to the "af-

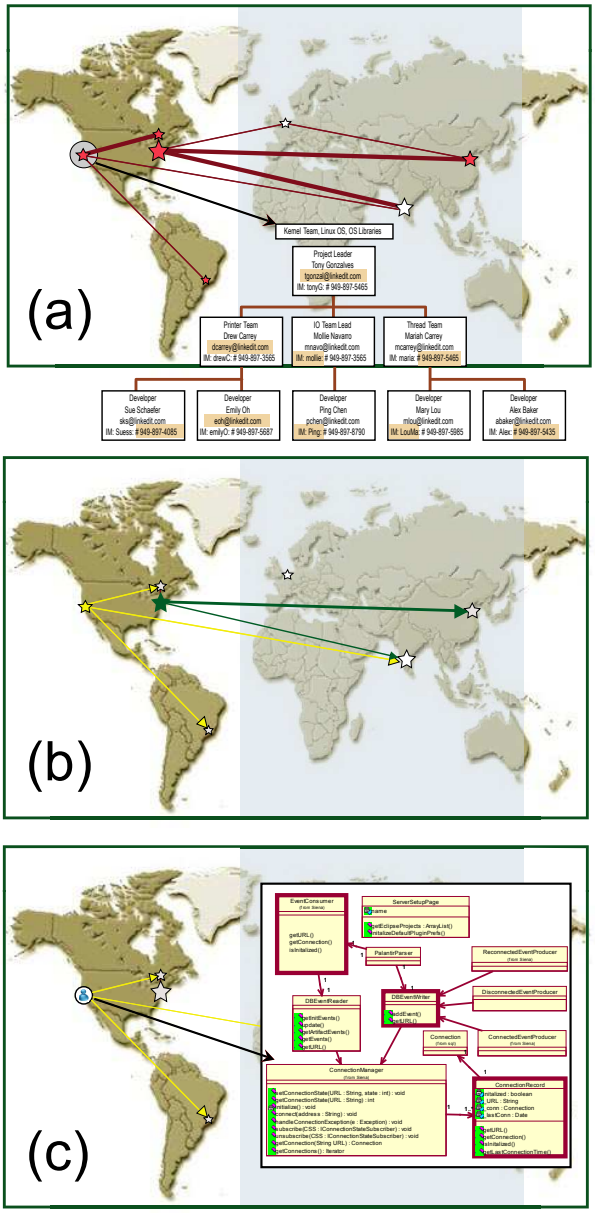

Figure 1. Alternative forms of the World View.

fected" team that they will have to resolve these conflicts before their code can be integrated.

3.3. Information Visualization

Strategy 3: Design specialized visualizations geared towards awareness in the large.

In order to avoid excessive context switching or cognitively overloading the user, most coordination tools present awareness information as contextualized icons (annotations on artifacts that have undergone changes) within the development environment. In multi-team development, developers rarely check-out code from other teams into their workspaces making these kinds of representation unsuitable. Moreover, the lack of real estate available in such visualizations severely limits the extent of information that can be displayed. Providing a comprehensive view of multi-team interactions (e.g., inter-team interactions, interdependencies on shared artifacts, effects of changes to these artifacts) is extremely difficult via these displays.

Specialized visualizations that concisely display the dynamics of the entire project (e.g., location of teams, shared artifacts and dependencies of teams around these artifacts, affected teams due to changes to a shared artifact) are needed for awareness in the large. We believe that such specialized visualizations complement the regular development-environment centric displays and that developers should be warned of potential conflicts, both arising from within and outside of the team, through the latter.

4. Illustration of Strategies

In this section, we present the design of a new view (the World View) that we intend to build for our workspace awareness tool, Palantír (see [14] for further details). The World View draws on our aforementioned strategies and builds upon Palantír's infrastructure of tracking development activities through developer interactions with CM systems.

The World View (see Figure 1a) provides a comprehensive view of the team dynamics of a project, regarding the geographical location of teams, the time zones of their operations, and the interdependencies among teams. This view is intended to help developers involved in global software development answer questions such as, who are my extended team members, what is the interrelation between our teams, how should I contact other team members?

In this view, teams are represented as "red stars" on a world map and interdependencies among teams are shown as "lines" connecting them. The size of the star denotes the size of the team; larger teams are represented as larger stars. Interdependencies among teams are determined based on the number of shared artifacts, which are identified through program analysis of the code base. The thickness of the lines represents the extent of sharing: the thicker the lines, the larger the number of shared artifacts. Through this view, developers can discern at-a-glance which teams are tightly-coupled and through which artifacts (mouse-hovers display the list of shared artifacts).

Right clicking on a team icon displays the organizational structure of that team (see inset in Figure 1a).

Each chart is annotated with project details of that specific team (e.g., the team name, the product feature under development, the list of shared artifacts). Other nodes of the chart display the names of developers in the team, their contact information, and their preferred mode of communication (via highlights in the chart).

Shaded areas of the map represent countries where it is dark. Teams that are still active in the shaded area of the view (based on development activities monitored via CM workspaces) are shown as "white stars". Identifcation of active teams (or active developer in the organizational chart) helps a developer seeking assistance choose the right mode of communication (e.g., email, instant messaging, telephone).

Coordination of changes to shared artifacts is essential to multi-team development, since these teams generally perform implementation tasks in relative isolation. Due to this isolation, teams largely remain unaware of changes to shared artifacts unless they are directly informed or their code fails to integrate. An alternate form of the World View displays the impact of a specific change on teams (see Figure 1b). In this view, arrows represent the direction of conflicts (changes performed by which team affects which team) and the thickness of the lines denotes the extent of the conflict: the thicker the line, the larger the significance of the conflict. Here, significance is calculated as the number of artifacts that are affected by the change. Teams and their respective "arrows" are color coded to differentiate conflicts arising from different teams. This view can also be configured for the individual developer to show which changes by a specific developer affects other teams (see Figure 1c). The artifacts responsible for the conflicts are highlighted in red (see inset in Figure 1c).

5. Conclusions

Awareness of inter-team development activities is a promising means of understanding and coordinating changes to shared artifacts and is particularly useful for global software development since one of the primary problems faced by global development is the coordination of changes to shared artifacts and interfaces. Current awareness-based coordination tools are geared towards small distributed teams and cannot handle the large scale of operations involved in global development. In this paper, we present three strategies, namely information abstraction, impact analysis, and specialized visualizations through which awareness tools can handle coordination of multiple teams. We also present a new view for Palantír that better addresses inter-team awareness by using the aforementioned strategies.

6. References

1. Appelt, W. *WWW Based Collaboration with the BSCW System*. Proceedings of *Conference on Current Trends in Theory and Informatics*. 1999: p. 66-78.
2. Berliner, B. *CVS II: Parallelizing Software Development*. Proceedings of *USENIX Winter 1990 Technical Conference*. 1990: p. 341-352.
3. Cheng, L.-T., et al., *Building Collaboration into IDEs. Edit -> Compile -> Run -> Debug -> Collaborate? ACM Queue*. 2003. p. 40-50.
4. De Souza, C.R.B., et al. *Sometimes You Need to See Through Walls - A Field Study of Application Programming Interfaces*. Proceedings of *Computer-Supported Cooperative Work*. 2004. Chicago, IL: p.63-71
5. Dourish, P. and V. Bellotti. *Awareness and Coordination in Shared Workspaces*. Proceedings of *ACM Conference on Computer-Supported Cooperative Work*. 1992. Monterey, CA, USA: p. 107-114.
6. Grinter, R.E. *Recomposition: Putting It All Back Together Again*. Proceedings of *ACM conference on Computer supported cooperative work*. 1998. Seattle, Washington, USA: p.393-402.
7. Herbsleb, J., et al. *Introducing Instant Messaging and Chat in the Workplace*. Proceedings of *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*. 2002. Minneapolis, Minnesota, USA: p. 171-178.
8. Herbsleb, J. and R.E. Grinter. *Splitting the Organization and Integrating the Code: Conway's law revisited. Proceedings of the 21st international conference on Software engineering*. 1999. Los Angeles, CA, USA: p. 85-95.
9. Herbsleb, J.D. and R.E. Grinter, *Architectures, Coordination, and Distance: Conway's Law and Beyond.* IEEE Software, 1999: p. 63-70.
10. Magnusson, B. and U. Asklund. *Fine Grained Version Control of Configurations in COOP/Orm*. Proceedings of *Sixth International Workshop on Software Configuration Management*. 1996: p. 31-48.
11. Nardi, B., S. Whittaker, and E. Bradner. *Interaction and Outeraction: Instant Messaging in Action*. Proceedings of *Computer Supported Cooperative Work*. 2000. Philadelphia, PA: p.79-88.
12. O'Reilly, C., P. Morrow, and D. Bustard. *Improving Conflict Detection in Optimistic Concurrency Control Models*. *Proceedings of the Eleventh International Workshop on Software Configuration Management*. 2003. Portland, Oregon: p. 191-205.
13. Olson, G. and S. Teasley. *Groupware in the Wild: Lessons Learned from a Year of Virtual Collocation*. Proceedings of *ACM conference on Computer Supported Cooperative Work*. 1996: p. 419-427. ACM Press.
14. Sarma, A., Z. Noroozi, and A. van der Hoek. *Palantír: Raising Awareness among Configuration Management Workspaces*. Proceedings of *Twenty fifth International Conference on Software Engineering*. 2003. Portland, Oregon, USA: p. 444-454.