An Environment for Synchronous Software Development

Lile Hattori and Michele Lanza
REVEAL@ Faculty of Informatics - University of Lugano, Switzerland

Abstract

Collaboration is essential for the development of complex software systems. When a team of developers is geographically distributed, collaboration aspects –such as awareness, communication and synchronization– are compromised by physical distance.

We present an approach, named synchronous development*, to reduce the negative distance effects on collaboration. We use a fine-grained change tracking mechanism and instantaneously notify any developer working on the system about changes being performed by other developers. We believe that our approach can augment each developer's knowledge of the project and, consequently, promote interaction and increase productivity.*

1 Introduction

Collaborative software development involves a team of developers working together to produce software systems. When a team is geographically distributed, collaboration aspects, such as awareness, communication, and synchronization are compromised by distance. Sangwan *et al.* suggest that once a team is separated by more than 50 meters further distance becomes immaterial [15]. Thus, regardless of whether a wall or an ocean separates developers, there is a need for solutions to minimize the negative effects of distance on collaboration. Moreover, Herbsleb *et al.* assessed that when a team is not sharing a single room, and face-to-face communication is lost, the willingness of developers to help others and the ability to spot specialists drops dramatically [6].

Indeed, the informal interaction that happens within co-located teams plays an important role on the coordination of the activities and on individual awareness. We consider awareness as an understanding of the activities of others, providing a context for one's activities [1], and also leading to an increased "habitability" of the code [3]. A study conducted by Herbsleb *et al.* evidenced that the decrease on communication and awareness of cross-site teams causes delays when compared with same-site work [6].

An equally important aspect of collaborative development, which is severely compromised by distance, is code synchronization. Currently software repositories are still the only *de facto* synchronization point. This inevitably introduces a latency for the propagation of changes made by developers, also because of the check out/check in model that software configuration management (SCM) tools use: only when a developer decides to commit the changes to the repository will they become visible to the others. The lack of awareness combined with such an outdated synchronization model is a potential generator of merge conflicts. This situation encourages developers to rush to check in their changes, so they are not the ones to deal with merging [4], thus directly affecting code quality.

Tool support for remote collaboration is able to augment the awareness of developers and can facilitate the propagation of changes. There are a few noteworthy recent efforts in this context, such as IBM's Jazz.net platform [1] or Microsoft's CollabVS system [5], which are able reduce the synchronization latency by propagating changes at file/class level immediately after they happen. Concurrently, a number of academic researchers have investigated how the information produced by integrated development environments while one is developing can be exploited. Noteworthy efforts in this context are the Mylyn tool [7] and Robbes' Sypware tool [13] which records every single change performed by a developer. Robbes proved that such an approach can produce information of unprecedented quality that can be used for both maintenance and forward engineering activities [10]. However, Spyware only collects the changes of a single developer and is unable to handle simultaneous changes made by a group of developers.

Our goal is to bring Spyware's change-centric approach into a collaborative context, i.e., to record every change made by a group of developers while they are programming and making them aware of what everybody is doing before the changes are checked into the repository. The immediate propagation of small changes can have a series of positive effects on collaboration, e.g., reducing duplication of work, preventing merge conflicts, and in general providing a more holistic development experience to developers.

[1] See `jazz.net`

We have started a prototype tool implementation, called Syde, which enriches the Eclipse IDE to create an environment for synchronous development. Our goal is not to replace existing SCM systems, but to complement them by allowing each developer of a team to know who is changing which elements of a program in *real time*.

We briefly discuss related work and then detail the roadmap that we plan to pursue to attain our goal of a synchronous software development environment.

2 Related Work

Tool support for collaborative development ranges from full-fledged platforms to solutions for specific problems.

In one extreme is the Jazz.net platform, built on top of Eclipse, a commercial environment that supports collaborative development processes and also incorporates its own repository [1]. Jazz.net is designed to be the central tool for planning and managing development activities, which can overwhelm the developers environment with unrelated tabs, views and text, and distract them from coding. Microsoft's CollabVS extends the Visual Studio programming environment by adding collaboration channels, such as text and audio-video chat, browsing of remote unchecked versions of files, and notification of presence in elements inside a file [5]. Although CollabVS targets collaboration among developers, it still relies on the classical check out/check in model and treats files as the lowest level of granularity.

In the other extreme there is a number of valuable efforts for solving specific collaboration issues. Palantir is an Eclipse plug-in that addresses direct and indirect conflict merging, where direct conflicts are caused by concurrent changes to the same artifact and indirect ones are caused by changes in one artifact that affect concurrent changes in another artifact [16]. Schneider *et al.* use a shadow CVS repository to record changes every time that someone edited a file [17]. The shadow repository is then mined and information about who is working with what is visually presented to developers with the aim of augmenting group awareness. The Emergent Expertise Locator is a Jazz.net plug-in that recommends experts according to developers' communication and project history. It essentially extracts emergent teams from the use of files by developers [9]. Mylyn is an Eclipse plug-in that creates a context for each task of a project by recording which files a developer edits while performing the task [7]. The task context helps developers to keep focus on their tasks, remember the classes related to an old task, and consequently improve productivity. A similar approach is NavTracks, which records the navigation of a developer to extract relationships among files [18], in order to recommend other files that might also be of interest to complete a task.

Outside the collaborative context, Spyware introduced the change-base approach to software evolution [14], in which every single change that a developer performs is saved. This valuable source of information was used to assist both reverse and forward engineering activities, such as the characterization of development sessions [11], and recommendation of code completion [12] . However, Spyware is restricted to a single-developer environment. We believe that applying Spyware's change-base approach for a collaborative environment can leverage both collaboration and software evolution analysis.

3 Synchronous Development

Our goal is to create an environment that supports synchronous development, thus enabling developers to know who is changing what immediately after changes on a system have been performed by anyone working on the system. To do so, we enrich the IDE with information about what the other developers are doing. To not disrupt the flow of programming, such information must be displayed in a non-intrusive way. Once notified, a developer can then decide whether to update to the newest version, even if it is not yet present in the project's version repository.

We envision a granularity of information beyond the commonly used files: A developer can set the desired information level, and can update to newer versions of fine-grained object-oriented entities like methods, classes, etc.

We started the implementation of *Syde*, a tool that offers a synchronous development environment to teams of developers. Syde is a client-server application in which the client is an Eclipse plug-in responsible for capturing and announcing changes, while the server collects, saves and distributes changes to all instances of the client.

Syde is not designed to replace versioning systems. Instead, it should be used as a complement. Syde runs concurrently with one versioning system's plug-in and should not interrupt or block its use, i.e., developers still have to check in their changes to the repository when they complete a task.

The architecture and information flow of Syde are illustrated in Figure 1. Syde currently features the following components:

- *The Inspector and the Collector.* Syde's inspector implements listeners to capture from Eclipse's workbench changes performed and classes viewed by a developer. The inspector collects *every* change that is happening in the IDE, similar to Robbes' Spyware tool [13]. More specifically, it records two distinct types of data: the actual changes; and metadata, which contains the authors name, a timestamp, a status of the change, and also the names of classes being viewed. Syde's collector receives information from the inspector and stores it in a centrally accessible repository. This data

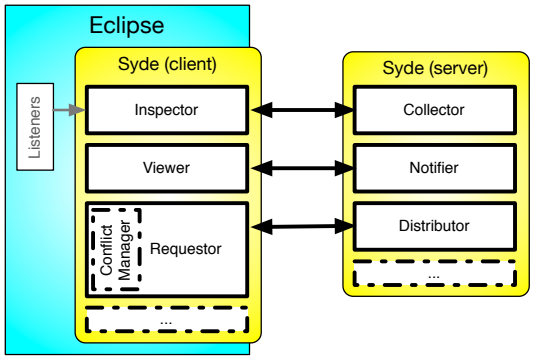

Figure 1. Syde Architecture

is being used by the notifier, but it can also be used to perform software evolution analysis.

- *The Notifier and the Viewer.* Syde's notifier maintains a list of client instances that need to be notified of any change, and is responsible for broadcasting the metadata to all members of the team. Syde's client features different ways to display information about the changing system within Eclipse itself, thus providing awareness of changes to all developers. In the future Syde should offer to developers additional means of augment awareness. One possibility is to allow developers to broadcast messages, e.g., if someone is having difficulties with a specific part of the code and does not know who to ask, he could broadcast for help.

- *The Distributor and the Requestor.* Once a developer has become aware that certain parts of the system have changed, he can preempt the underlying classical versioning system and request from the Syde server an update of specific parts of the code, which are then sent by Syde's distributor, and updated in the client's source base. In case there is a merging conflict between local and newer versions, the requestor module offers a conflict manager (not yet implemented), based on Fluri's ChangeDistiller algorithm [2].

Syde will provide extra information without disrupting or distracting developer from his work. Change alerts should be displayed as minimum markers beside each changed element, and in a view that the developer can simply close or minimize. Information about who is viewing what should only appear if and when the developer requests. Furthermore, Syde should be easily enabled and disabled.

4. Preliminary Results

We have started to use Syde to monitor and record developers' activities with the goal to understand how they behave when programming. This initial analysis will help us to adjust the granularity of change information that Syde will supply for teams of developers. Up to now, we are monitoring two Java projects, each with one developer involved: our Syde plug-in; and X-Porter, which is the model of a student's bachelor project, called X-Ray [8].

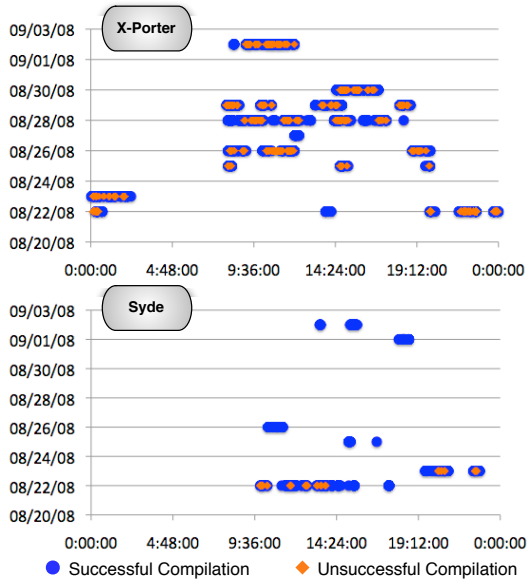

Figure 2. Preliminary results

We analyzed a total of 2,690 changes from X-Porter and 457 from Syde plug-in within a time span of 14 days. Figure 2 shows the results. During this period X-Porter was under development, while the main activity for Syde plug-in was bug fixing, a maintenance activity. In the graphs each mark corresponds to a file that a developer tried to compile at a certain time. The blue circles are files successfully compiled and the orange diamonds are files that contained compilation errors and were not saved as new versions.

While we collect more change information that what would be possible with a conventional versioning system, a valid counter-argument is that we also collect "noise", for example information about changes that would correspond more to a trial-and-error development style, and which are therefore not important for other developers. In the future we plan to implement a series of mechanisms to process the collected information. This first experiment we performed is to be considered as a proof-of-concept.

By comparing the two graphs, we see that when the main activity is development unsuccessful compilations occur much more often than during maintenance. A possible interpretation is that Java developers implement by trying, breaking and fixing their code. On the other hand, when fixing specific parts of the code, they tend to circumvent com-

pilation errors. For future analysis, we plan to investigate the possible existence of well defined patterns of breaking and fixing code that indicate more appropriate strategies of sharing changes.

5. Conclusion

We have presented initial work on a tool for synchronous development that addresses problems that are aggravated when developers are geographically distributed. The main focus of our tool is to minimize the negative effects of distance on the current synchronization model offered by software configuration management systems. Syde allows developers to know who is performing which change, be it a class or a method change, before it is checked in the repository. Each new change becomes instantly available for the team, which potentially reduces merge conflicts: two developers who are working on the same class are aware of this situation and can avoid editing the same method; or one developer can wait for the other to finish editing the class. However, if a merge conflict happens, Syde offers a conflict manager to help the developer to solve it.

We are at the early stages of our research endeavor: The next steps will be to increase the granularity of changes that are captured by the plug-in, to finish implementing the conflict manager, and to start recording the files that a developer only views during a development session. In addition, we envision the refinement of the visualization of change alerts, the implementation of an instant messaging service, and the use of visual metaphors to help the team to understand how they could improve their interaction. We plan to apply Syde to small teams for a pre-defined period, monitor the use, interview them to collect feedback, and use the historical data to perform evolutionary analysis.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science foundation for the project "REBASE" (SNF Project No. 115990).

References

[1] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 107–114. ACM, 1992.

[2] B. Fluri, M. Würsch, M. Pinzger, and H. Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Trans. Software Eng.*, 33(11):725–743, 2007.

[3] R. P. Gabriel. *Patterns of Software*. Oxford University Press, 1996.

[4] R. E. Grinter. Supporting articulation work using software configuration management systems. *Comput. Supported Coop. Work*, 5(4):447–465, 1996.

[5] R. Hegde and P. Dewan. Connecting programming environments to support ad-hoc collaboration. In *Proceedings of ASE 2008 (23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE CS Press, 2008.

[6] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. Distance, dependencies, and delay in a global collaboration. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 319–328. ACM, 2000.

[7] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 1–11, New York, NY, USA, 2006. ACM.

[8] J. Malnati. X-ray - an eclipse plug-in for software visualization. Bachelor's thesis, University of Lugano, June 2007.

[9] S. Minto and G. C. Murphy. Recommending emergent teams. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 5, Washington, DC, USA, 2007. IEEE Computer Society.

[10] R. Robbes. *Of Change and Software*. PhD thesis, University of Lugano, Switzerland, Dec. 2008.

[11] R. Robbes and M. Lanza. Characterizing and understanding development sessions. In *Proceedings of ICPC 2007 (15th International Conference on Program Comprehension)*, pages 155–164. IEEE CS Press, 2007.

[12] R. Robbes and M. Lanza. How program history can improve code completion. In *Proceedings of ASE 2008 (23rd ACM/IEEE International Conference on Automated Software Engineering)*. ACM Press, 2008.

[13] R. Robbes and M. Lanza. Spyware: A change-aware development toolset. In *Proceedings of ICSE 2008 (30th International Conference in Software Engineering)*, pages 847–850. ACM Press, 2008.

[14] R. Robbes, M. Lanza, and M. Lungu. An approach to software evolution based on semantic change. In *Proceedings of FASE 2007 (10th International Conference on Fundamental Approaches to Software Engineering)*, pages 27–41, 2007.

[15] R. Sangwan, M. Bass, N. Mullick, D. J. Paulish, and J. Kazmeier. *Global Software Development Handbook (Auerbach Series on Applied Software Engineering Series)*. Auerbach Publications, Boston, MA, USA, 2006.

[16] A. Sarma, G. Bortis, and A. van der Hoek. Towards supporting awareness of indirect conflicts across software configuration management workspaces. In *Proceedings of ASE 2007 (22nd IEEE/ACM International Conference on Automated Software Engineering*, pages 94–103. ACM, 2007.

[17] K. A. Schneider, C. Gutwin, R. Penner, and D. Paquette. Mining a software developers local interaction history. In *MSR '04 : Proceedings of the 1st International Workshop on Mining Software Repositories*, pages 106–110, 2004.

[18] J. Singer, R. Elves, and M.-A. Storey. Navtracks: Supporting navigation in software. In *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, pages 173–175, Washington, DC, USA, 2005. IEEE Computer Society.