In Depth Study EDA270, Coaching of Programming Teams

Team awareness tool support for concurrent development in XP

D01, Mathias Bruce (`d01mbr@student.lth.se`)
D01, André Johansson (`d01aj@student.lth.se`)

Lund Institute of Technology

February 20, 2007

Abstract

This in depth study concerns the problem of insufficient team awareness during concurrent development in eXtreme Programming. We will concentrate on the problems that arise around concurrent development on shared resources without proper communication as part of the sharing process. A solution proposal with a concrete implementation will be presented, together with a comparison of existing tools and and small scale test trial of the implemented tool.

Contents

1 Introduction

XP is an interesting development methodology, not the least when it comes to team communication and awareness. Planning games, informal stand up meetings and many other disciplines highly encourage constant communication between team members. As a result of this, every team member *should* be aware of the project architecture, current development status and many other properties. This is very valuable, as communication helps avoid many unnecessary mistakes and misunderstandings.

XP dictates that the implementation of product features should be split into stories - and large stories into tasks - which in turn are assigned to individual pairs of developers. This ensures that no two pairs are simultaneously implementing the same feature. Collective code ownership can, however, easily lead to two or more pairs simultaneously working on the same resources. Concurrent development of the same resources might result in a Software Configuration Management (SCM) merge conflict that must be manually resolved, meanwhile halting team development and requiring debate about whose work should be scrapped and whose should be kept, etc. In [5], conclusions are also drawn that XP team sizes suffer from poor scalability due to the increase of communication overhead that comes with larger teams.

Solving this problem with tool support for awareness has been widely discussed, both as stand alone tools and as extensions to existing groupware applications. We also propose such a solution, and have developed a tool for this as a plug-in for the popular open source Integrated Development Environment (IDE) Eclipse. While mainly an IDE and not a groupware application in the traditional sense of the word, Eclipse's open platform and extensible plug-in infrastructure has spurred the creation of numerous team collaboration extensions including (but certainly not limited to) SCM plug-ins. Our solution will make use of existing SCM plug-ins and attempt to extend their usefulness.

To ascertain the usefulness of the tool, we conducted a test trial to collect user experiences. The trial was carried out with participants from a course in eXtreme Programming at Lund Institute of Technology. Since the development in this course is carried out in the Eclipse IDE, our proposed solution was implemented as an Eclipse plug-in.

Having described the problem background above, we will continue by defining the goals of our attempt at remedying it. The goals of a process improvement tool that adds any complexity to the situation must also explicitly take into account the apparent risks of process impediment. We will then describe our chosen solution for the problem and motivate our choices. A short summary of experiences from a testing trial will then follow, where we will discuss positive and negative experiences encountered while evaluating the tool. Finally, we will briefly discuss existing tools, possible future expansions and our conclusions of this study.

2 Goals

An obvious goal of such a tool is to provide the missing information that constitutes the gap in communication we described in the introduction. All the essential information necessary to solve the awareness problem must be communicated, but no other.

The tool should work with real time user notification. XP is an agile methodology, and large overhauling changes can be decided on and carried out quickly. No Change Control Boards will convene and no other special action will be taken that might indirectly notify a user of what's going on, save for normal team communication. The team members must communicate and coordinate such information continuously, and that is what we want to help them do. As the awareness concept is discussed in [2], a difference is made between proper "focused collaboration" and having only "peripheral awareness" of other team members' actions. We wish to provide the former, as the latter runs risks of exposure to the problems we wish to avoid.

Providing functionality that solves the described problem is not enough. In designing such a tool one must also be wary of the inherent risk that adding further information to an already complex environment might only add confusion. IDE's as a software type are often advanced and featureful, which in turn affects it's user interface. Eclipse is no exception, and we did not want our tool to intrude unnecessarily into the users workspace, nor distract him/her from the task at hand. We wanted to keep it visually small on non-distracting, but noticeable enough so as not to be forgotten by the developer. In [3], previous findings are revisited that define awareness as "an understanding of the activities of others, which provides a context for your own activity", and also argue that "awareness information should be passively collected and distributed rather than explicitly provided by participants, and it should be presented in the same shared work space as the object of collaboration".

As something that in part follows from having a simple user interface, we wanted the tool to be very easy to use. It should be simple to configure and have as few controls as possible, so that the tool can work *for* the user, instead of the user working *with* the tool.

We wanted the tool to be lightweight, in all aspects. It should occupy little disk space, consume few cpu cycles and use little network bandwidth. Some tools might be suffered by developers to consume a desktop computers full abilities, but we suspect this would not be one of them.

Portability is an important issue. If portability between operating systems could be achieved, it would be a valuable feature.

3 Existing tools

Functionality to make teams aware of what everyone currently is working on seemed like an idea someone already would have implemented, however after searching for such a tool we could not find such a tool that worked in a realtime context.

Most tools seems to be working reactive, that is when something have gone wrong. Other tools rely on non-direct methods of communicating such as email lists, text chat and version control systems logs or other kinds of comment logs [1]. This is much too slow in our opinion, a developer needs notification immediately when there is an imminent collision.

We have not been able to examine larger enterprise systems and it's usually quite hard to deduce from these systems if they contain a component that does what we want or not. One reason is that they are often very feature rich and it's hard to spot a function if you don't have thorough experience with the system, another is simply that there is usually a lot of buzzwords to filter out and little examples and/or fact in the available presentations of such systems.

Since we've chosen to build an Eclipse plugin we've tried to find other plugins or official Eclipse projects that does what we want but we haven't found any. The closest plugins (judging from their descriptions) are Doggone and Sobalipse. These projects seems to be dead (development is halted) and non existent anymore. The descriptions of the plugins are taken directly from the Eclipse plugin repository.

- **Moomba** - See subsection 3.1
- **Doggone** - A notification and control plugin for Eclipse that helps distributed teams avoid clashes when committing to a common source repository.
- **Sobalipse** - Sobalipse is an Eclipse plugin, which realizes realtime collaboration. Sobalipse enables you to do work with remote users, such as, tele-pair programming, realtime code reviewing, and so on.

3.1 Moomba

The closest competitor to our tool is Moomba [6]. Moomba is a whole suit of tools designed for distributed development in general with special attention to Distributed eXtreme Programming. Moomba defines a sound three tier theoretical model for awareness and collaboration which is later used to support a tool implementation.

Unfortunately we haven't managed to test the tool first hand (we do not even know if it's free too use). However from the exhaustive description in Reeves et al.'s [6] article, we've concluded what we believe are some drawbacks of the tool.

- Moomba is a large environment containing both web applications and an own IDE for development. This seems to us to be contradictory what Reeves et al. mentions in the beginning of their article that awareness tools should be integrated into existing systems. Here a developer would have to change tools completely.

- Moombas size and feature richness probably makes the learning curve steep and hence takes a step away from XP's simple ideas.

Moomba is widely larger than our little tool and we belive it is a useful environment. However our tool aims to be a smoother aid for the developer than a full blown IDE. Using the widely appreciated and very extensible Eclipse IDE as base we do not lock ourself to some specific language environment nor operating system. We also view the size of our project as an advantage - the fewer buttons to press, the easier to use.

4 Solution

The name of the tool we propose as a solution is ”Stenkoll”. It is a swedish slang word of sorts, conveying the feeling of ”staying on top of the game”, knowing the status of things surrounding oneself.

A development project can contain any type of shared resource, but our tool only aids collaboration when developing text based resources such as source code or other text documents.

The solution consists of a client-server architecture. The client is a lightweight Eclipse plug-in that collects information about the users actions. Each users client in turn connects over a TCP/IP network to a server that synchronizes all the information and keeps all clients aware of each others actions. Notifications are therefore delivered in real-time and as soon as the potentially dangerous situation arises, which allows for quick discovery and countermeasures before any damage is done.

Information about the local status of a client is harvested not only from Eclipse itself, but also from metadata stored on local disk by the Eclipse SCM plug-in. Currently (for the sake of the trial we wished to conduct) the tool only supports Concurrent Versions System (CVS), but could easily be extended to support Subversion (SVN) or any other SCM tool that stores metadata on local disk regarding local modification status and local revision number. As the metadata is read from disk, the SCM tool must not necessarily be related to Eclipse in any way.

4.1 Dangerous situations

The tool concentrates on three situations in which we feel it is necessary to alert the user. They are as follows:

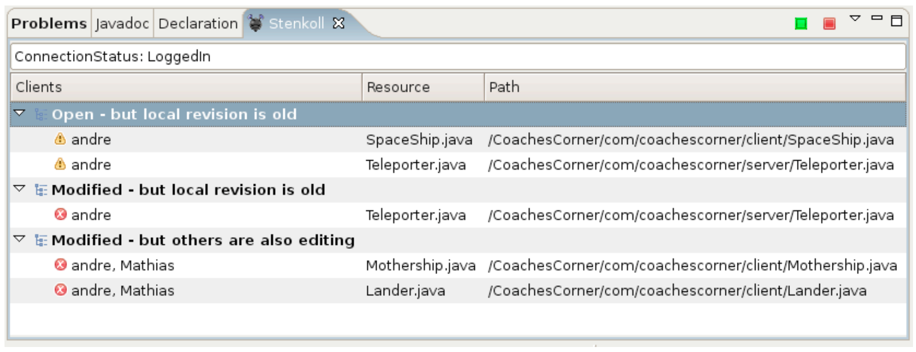
1. The user has a file open, but the file's local revision number is older than the revision in the SCM repository.
2. The user has locally modified a file, but the file's local revision number is older than the revision in the SCM repository.
3. The user has locally modified a file which is up to date with the SCM repository, but atleast one other user is also locally modifying the same revision of that file.

The problems that might arise and could unneccesarily hinder development are easily solved. In situation 1), the user should simply update his/her file to the latest revision in the SCM repository to make sure he/she has accurate information at hand. In situation 2), the user should also update his/her file to the latest revision in the SCM repository and resolve any SCM merge issues that might arise. In situation 3), the user should at once contact the other user(s) and coordinate their work to avoid future merge conflicts and/or duplicated work efforts.

If they are in the same room they might go talk to each other face to face to synchronize the development in a safe manner. If they are geographically distributed they might discuss the matter using instant messaging or VoIP. Whatever their means of coordination, this tool simply aims at raising awareness of possibly colliding work, thereby avoiding e.g. SCM merge conflicts and/or duplicated developments efforts.

4.2 User Interface

The clients (Eclipse plug-in) canvas for presenting the user with information is a tab in the lower pane of the Eclipse IDE, residing next to common tabs as Problems, Task List and Console output (figure 1). This is a fairly small and unintrusive User Interface (UI) widget. Only two controls are present, a connect button and a disconnect button to control server connectivity.



ConnectionStatus: LoggedIn

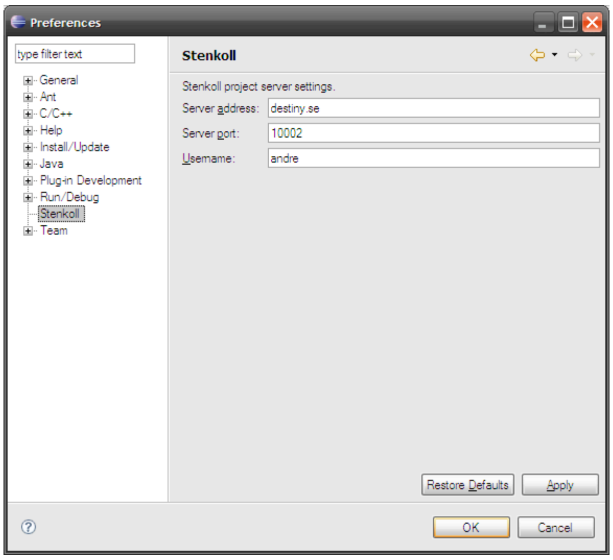
Clients	Resource	Path
Open - but local revision is old		
andre	SpaceShip.java	/CoachesCorner/com/coachescorner/client/SpaceShip.java
andre	Teleporter.java	/CoachesCorner/com/coachescorner/server/Teleporter.java
Modified - but local revision is old		
andre	Teleporter.java	/CoachesCorner/com/coachescorner/server/Teleporter.java
Modified - but others are also editing		
andre, Mathias	Mothership.java	/CoachesCorner/com/coachescorner/client/Mothership.java
andre, Mathias	Lander.java	/CoachesCorner/com/coachescorner/client/Lander.java

Figure 1: Stenkoll view

4.3 Configuration

Configuration is equally simple. Amoung the common Eclipse preferences the user finds a section named after the plug-in, Stenkoll (figure 2). Options requiring configuration are username describing the user and server address and port to connect to.

Finally, we consider the goal of portability to be sufficiently achieved as Eclipse itself is portable to largest development platforms at the time of writing.



Stenkoll

Stenkoll project server settings.

Server address: destiny.se

Server port: 10002

Username: andre

Figure 2: Stenkoll preferences page

5 Initial testing experiences

In order to ascertain the usefulness of such a communication tool, our original intention was to conduct a test trial with an experiment comparing certain aspects of the development process between development teams using the tool and teams not using it. The teams consisted of computer science students participating in courses we will briefly describe below. In our attempts at orchestrating the trials we encountered difficulties which we feel lend yet another hand to the argument that such a tool must be easy to use and unintrusive to be able to attract users.

5.1 XP Course at LTH

The XP projects at LTH are a result of two courses, one mandatory for the masters students of Computer Science and one optional. The mandatory course gives an insight into XP methodology through team development and is one of the first courses the students have which primarily focuses on teamwork. The optional course focuses on educating students in pairwise coaching of the aforementioned teams, consisting of about 10 persons each [4].

5.2 Hesitation regarding Yet Another Tool

At first when we presented our tool to our test subjects we encountered some hesitation to use yet another tool. The students in the course are introduced to many new processes and tools in a quick fashion which explains the hesitation. After a demonstration and some initial testing our team accepted the plugin and thought the idea and layout was good. A number of minor annoyances were corrected (such as the tool would notify when not actually critically needed). Spreading the tool to other teams was a bit harder. A couple of developers accepted the solution after a demo, but in general they felt to stressed up to start using the tool.

Many development tools are advanced by nature. They are also designed for advanced users, but that shouldn't implicitly mean that you can ignore the ease-of-use aspects when designing them.

6 Future expansions

The tool has some known limitations that would be interesting to add in a future version, of which we will mention a few here.

Branch support is lacking. If a user branches the resources and continue working in the side branch, he/she should not be connected to the same server as his/her team members still working on the main branch.

On-the-fly merge testing would also be very valuable. If the tool discovers possible merge problems, it could simulate a resource merge using the same algorithms as the underlying SCM tool. Thereby, the tool could tell the user if they are risking a merge conflict that must be resolved manually or one that the SCM can resolve automatically.

The tool does not **really** know which version is the latest in the SCM repository, as this would require continuously polling the repository server. We wanted to keep the tool event driven, and therefore it only knows of the latest version that any connected team member currently has. This, however, is not a problem if all team members are connected to Stenkoll while working. As soon as a client commits a new revision, the tool will discover it locally and notify the server (which in turn propagates the information to all the other clients).

Our test results are highly subjective and we would want to test the tool on larger teams (our team consisted of 8 developers) and on teams which aren't located in same geographical location (such as many open source teams).

To increase the validity of such a test it could also be performed on a large enough base of projects to get a statistical significance to prove or disprove our hypothesis of the tools usability.

7 Conclusions

As few were swayed to use the tool extensively, our conclusions will not be drawn from statistically secured data comparing teams with different prerequisites (using our tool or not), but from user experiences we gathered from our own team (that used the tool in it's entirety) and other teams in which some members used it.

The users responded well to the tool and claim they avoided some conflicts that could otherwise have slowed them down significantly, however a great need still exists for developers to learn the different processes of XP such as continious integration and updating regulary. Everyone seemed to approve of the design of the user interface and controls. Some commented on forgetting to use the tool due to it not always being visible on screen, and some would see its warnings but sometimes disregard them anyway. In the end the tool is merely an aid, and the processes are still the ones of greater importance. We believe, however, that staunch followers of XP's best practises would benefit from using our tool, and it should allow them the scale their teams slightly larger if necessary without exposing themselves to the problems described in this article.

8 Acknowledgements

Thanks to Görel Hedin and Lars Bendix for valuable input on article scope and tool trial realization.

Special thanks also to the eXtreme Programming team that participated in our trial: Nezdet Arif, Andreas Bergström, Mikael Brattmo, Brice Jaglin, Jonatan Jonsson, Per Magnusson, Christoffer Sawicki and Samuel Skånberg.

References

[1] Kevin Schneider Andrew Sutherland, Tadeusz Stach and Carl Gutwin. Architecture to support team awareness in large-scale software development. *CSCW*, 2006.

[2] Ronald M. Baecker, Dimitrios Nastos, Ilona R. Posner, and Kelly L. Mawby. *The user-centred iterative design of collaborative writing software.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[3] Carl Gutwin, Mark Roseman, and Saul Greenberg. A usability study of awareness widgets in a shared workspace groupware system. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 258–267, New York, NY, USA, 1996. ACM Press.

[4] Boris Magnusson. Course description for eda260: Software development in teams. *http://www.ka.lth.se/kursplaner/2006 eng/EDA260.html*, 2006.

[5] M. Muller and W. Tichy. Case study: extreme programming in a university environment. *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, pages 537–544, 2001.

[6] Michael Reeves and Jihan Zhu. An awareness model for supporting collaboration in distributed extreme programming. In *Software Engineering Research and Practice*, pages 468–476, 2004.