Codebreaker: decentralized, cooperative and flexible support for extreme programming software development

Baloian, N.[2,3], Konow, R.[1], Claude, F. [2], Tala, C.[1]

[1]Universidad Diego Portales , Ejército 441, Santiago, Chile
{rkonowkr,ctalasa}@al.udp.cl
[2] Universidad de Chile, Departamento de Ciencias de la Computacion, Blanco Encalada 2120, Santiago, Chile
{nbaloian,fclaude}@dcc.uchile.cl
[3]GITS, Waseda University, 1011 Okuboyama, Nishi-Tomida, Honjo-Shi, Saitama 367-0035, Japan

Abstract. This paper presents a system called CodeBreaker for supporting small and medium size software development based on an extreme programming principle. The system follows a decentralized model of development, which means, it does not requires a central repository. A set of rules for code ownership maintains the synchronization of the work among all members of the deveoping team which can wokl on- or offline. It allows fine-grained locking of parts of the code.

1 Introduction

The development of systems for supporting distributed programming teams has attracted the attention of many authors in the past. Most of these systems are developed for supporting a particular software development style. For example, in [1] a system for supporting distributed teams in extreme programming is presented. In [2] the authors describe a system for supporting distributed software development based on a peer-to-peer architecture, in opposition to the most common centralized repository architecture. Version control systems like CVS [3] or SourceForge [4] are perhaps the most frequently used today for supporting collaborative, distributed programming. Although the development of this kind of systems has been very prolific in the past, there are many reasons to believe that there is still room for improving software development support and that the last word is far from being said. This is especially true when we consider new situations that arise from new scenarios created by the existence of pervasive computing enabled by mobile technology like wireless LANs and smaller, lighter and more powerful notebooks. This scenario promotes the emergence of small programming teams, which may start developing a small to medium size project in a brain-storming like meeting. Such kind of situations is becoming more common, as most people use their own notebook computer as their

working machine anywhere, be it at home, at work, or even during a coffee break.

To more concretely illustrate the requirements of the software developing scenario we want to support, lets take the example of two or three programmers that get together and they exchange ideas about a new system they have just conceived in a planned or spontaneous brain storming session. They open their laptops and start developing a new project, which they more or less outline by creating new classes which contain just some sample code or comments. A wireless network may be present, permitting them to work synchronously. If not, they will have to exchange files to merge their work asynchronously, maybe by email or by pen drives. They decide to continue separately divining the job and responsibilities. All or some of the original members may meet again, new members who joined the project may be also present and they will have to merge their work. They will certainly welcome a tool for coordinating their work satisfying the following requirements:

i. **Work on a peer-to-peer architecture without having a central repository.** As we want to support people who may start a new development without previous preparation, a central repository may not be always available for all members at that moment. Because of this, every member of the developing group should have a copy of the project as updated as possible, even when working alone.
ii. **Allow synchronous and asynchronous collaborative working.** Of course the system should support the synchronous collaboration work when two or more users are on line, providing the adequate tools. But it should also allow to synchronize the work with other participants which are offline in the best possible way, and provide mechanisms for merging the code developed off line.
iii. **Allow the inclusion of new unforeseen participants** Because the system is aimed to support flexible and changing teams, there should be a way to include unforeseen participants and assign them tasks. However, the system should avoid an uncontrolled explosion of participants and maintain the order in the versioning of the code.
iv. **Allow fine grained and logical oriented locking of code.** In a less formal and flexible programmers team everyone may have access to all the code and be able to modify it. However, this condition may introduce too much complexity for synchronizing the work. A good trade-off solution may be that the system should give the possibility of locking finer parts of the code inside a file, like a procedure or even some lines inside a procedure. It also should allow to reserve names for allowing the locking of still not written, since participants may agree to distribute the work by just locking names of classes, methods or even variables which are still not written or used.

Of course, for this scenario we have an extreme programming style of development in mind. Extreme Programming (XP) is a software development methodology, that emphasizes bringing the project to the beta testing phase as quick as possible, reducing the time of planning phase and increasing the priority for the beta testing phase [5]. Currently, the cost and time to develop a small or medium-size software using the classic software engineering methods is too high.

XP stresses the collaborative work and distributed programming. Most of the systems claiming to support software development according to XP are focused on supporting the synchronous work. Some of them are mere collaborative editors while others include support for coordinating the work, like awareness and versioning mechanisms. Some authors have already pointed out to the necessity of not having a centralized repository to coordinate the work of a software developing team [6], while others also have stressed the necessity of having a fine grained, logical oriented locking of the code [7]. The decentralized model is without question the most flexible and most accepted model for the requirements of a XP project.

However, there is still no system which meets all the requirements mentioned. Developing such a system represents a challenge of high complexity, in the design and in its implementation. In this work we will present a system called Codebreaker for supporting small and medium size software development based on an extreme programming principle, meeting the requirements mentioned above. Because of the logical locking of the code requirement we will develop it for supporting a particular programming language which is Java. However, most of the restrictions that this language imposes are easily transferable when implementing the same system for any other object oriented language.

2 Related Work

Back in the late 80'and early 90's when the Internet was rapidly expanding, there was a great interest in the distributed systems. It was then predicted that such systems will be the dominant technology for the synchronous collaborative work in the future [8].We can nowadays confirm those predictions and add that these system have also deeply influenced the working style in all fields, Of course, computer system programming being was one of the first, and many systems have been developed since very early. We can classify those systems in two categories according to the aspect they stress with their support.

2.1 Versioning management systems

In the 1990's perhaps the most used tool for collaborative work synchronization was created, CVS, [3] initiating a wave of development of tools supporting Version Management. CVS problems are well known [9] : it uses a centralized model, a central data repository and only few operations or commands which can be executed offline. This makes this structure really unsuitable for synchronous collaborative programming development. All developers need access to the central server for almost all operations. Today, there is a whole family of CVS-like tools : GNU-Arch, Subversion, CSSC, PVCS , etc. These applications are frequently used in the Opensource community and also in large business environments. All of them follow the same schema: one central repository, and file-level permissions. (Check in, Check out). These tools are used for Version Management in mid to large software development projects with many programmers involved.

2.2 Collaborative development environments

One of the first approaches to the implementation of collaborative development environments is the Orwell system [10]. This system allows the Smalltalk programmers to develop programs using a common library. An interesting aspect of this system is that it organizes the developing system code in methods and classes instead of files, thus using a more logical approach to present the code. Another Collaborative Environment that follow the same idea of the Orwell system is Tukan [11].This synchronous distributed team programming environment for Smalltalk claims to solve the problems that Extreme Programming teams have. Tukan incorporates a version management system and adds awareness information, communication channels and synchronous collaboration mechanisms. It also provides a shared code repository with a distributed version management and the code integration can be made in a centralized or decentralized way. The IBM Rational ClearCase System [12] provides real time support for collaboration between developers located anywhere on the Internet. It uses a central server, that manages users permissions and differences between the source code versions. The server has also support for multiple repository server deployments for large-scale enterprise teams.

Another tool to which supports the collaborative editing of source code is the *Collab* add-on for the Netbeans 5.0 [13] . This add-on allows the NetBeans users to edit files collaboratively, share files and provides space to communicate with other developers.

3 The CodeBreaker

For the implementation of the CodeBreaker system there are mainly two central problems to solve: the the control of user rights to modify certain parts of the code (locking) and the merging of the works of all participants (merging), including the distribution of the information about changes made to the code, new participants and changes in the ownership of the code (which user has the lock for which part of the code) in a peer-to-peer synchronous/asynchronous scenario. Our approach to face these problems is based on implementing a set of rules and a system architecture that allows to implement them.

Since there are many good IDE for developing code in Java or other languages, it is better to use and extend one of them to include the missing functionalities instead of developing one from the scratches. CodeBreaker is developed as an extension of NetBeans [13]. The missing functionalities of this IDE are exactly those necessary for meeting the requirements described in points i to iv of chapter one. The rest of this chapter describes the mechanisms used to fulfill these requirements.

3.1 Rules for Code ownership

In order to allow the synchronization of the code being developed among the members of the group in an asynchronous scenario CodeBreaker imposes that any existing code in any of the participants' computer should be “owned” by someone. A

CodeBreaker code development project starts with one person defining the project an others joining it. Each new member including the one who created the project has to register an e-mail address and receives a digital signature. All members can develop new code which is owned by him/her. Other members will receive the code and can use, modify, and even share it with others, but the only "official" version can be distributed or approved by the owner. In this way, there will be always a final version of the entire software which will the sum of the code pieces each participant owes. the In order to allow users to delegate their work, users can pass the ownership of the code among each other. Figure 1 and Figure 2 show an example how ownership of code may develop during a project involving three programmers.

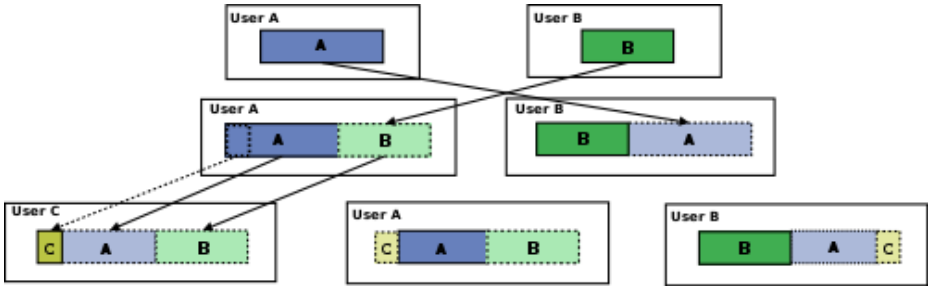

Figure 1: *Colors show ownership of the code: blue for user A, green for user B and yellow for user C.. In the first row, A and B start a new project writing both a part of the code. In the second, they merge their works and keep the ownership. In the third row, C joins the project and A grants ownership rights to part of the code.*

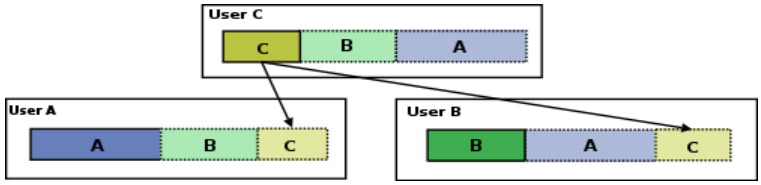

Figure 2: *User C works on the part of the owner code and distributes it to A and B with the new code included*

3.2 Exceptions to the Rules

It is important to maintain the rights of the owner of the code and the order of the project itself. It is also known that in many projects is impossible to maintain and respect every rule, so an alternative should exist. For example, it could happen that a certain user cannot work on the project anymore and that he is not reachable to ask him to delegate the work to other users. In this case there are two mechanisms that can be applied and the two can coexist giving more flexibility to the system. The first one is that a user can ask to overpass a rule giving the rest of the user of the system the responsibility of approving or rejecting the petition by voting. Such petition may be for getting the ownership of a certain part of the code or to force the acceptance of a

given modification.

3.3 Logical locking

As we already said, the entities of the code which can be owned are logical more than physical entities. Entities which can be locked are organized according to the JAVA organization of the code. The locking is done over a name of class, method inside a class or variable inside a method or class. In this way, it is possible to lock code which still not exists. The scope of the locking also follows the class hierarchy of Java: If a class name is locked (owned) all the extended classes will be locked as default. In the same way, if an interface file is owned, the implementations of those methods is also owned. This may be necessary in cases when one of the participants should write the same method for different classes, for example, the same programmer developing a drawing method for different objects representing graphical elements.

3.4 Synchronizing the work

Synchronization must be possible when working synchronously as well as asynchronously. When working synchronously the information about changes of any type is sent to all connected participants. When a latecomer joins a working session with one or more other participants, their records are compared to update information about changes. Only code changes which are issued by the owner of the code are forcibly exchanged so there is no conflict about which is the latest version, since the owner issues a correlative number when its code is ready to be distributed. This number is also used to check if the change has been incorporated already. When an owner wants to publish a new version of a code a file with an XML content containing metadata and data for the code is generated and signed with his digital signature. The same is done for distributing information about changes to the code ownership and new members.

For participants who are seldom online with the rest of the group or if various subgroups do not meet each other frequently CodeBreaker offers an asynchronous mechanism based on the use of e-mail. The XML files with the changes are sent to all email addresses of the project. Users can download them and process them offline.

As the system is supposed to work on an XP environment, the option of pair programming [14] is a very important issue. To allow pair programming, a user should ask for being watched by another user. The user that begins to watch should have permission of modifying parts of the source code and to see real-time the modifications made by the user that sent him the invitation. When both ended to work as a pair, the source code should be saved on both workstations, but the modification should be marked as from one user only, so that the owner receives only one confirmation of a given code.

3.5 Assigning Roles

CodeBreaker is aimed to support more a flat project structure in which every

participant has the same rights and responsibilities. However, sometimes even in small projects there may be a need of a certain hierarchy in order to maintain the synchronization among the participants. CodeBreaker introduces two mechanisms which allow this with flexibility. The first one is, when a user is created it may or not receive the right of accepting new participants. The number of participants which is allowed to invite can be also be set. The second one is while receiving the ownership of a code. The right to pass this ownership to a third can also be given or not.

4 The File System Architecture

For implementing the logical management of the code as described in chapter 3 CodeBreaker uses a three logical layer file system architecture as seen in figure 3. The bottom layer is the physical layer, containing the accepted java files. The middle layer is the metadata layer containing data for access management and presentation of the code. The upper layer is the logical file system which implements the emulated file system using the data stored in the other two layers.

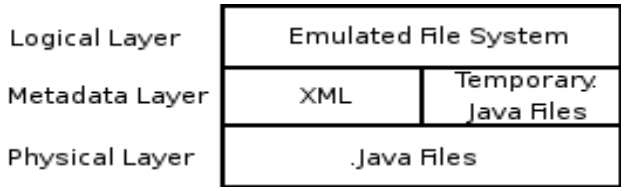

Figure 3: The three layer architecture of CodeBreaker

- Accepted Java Files : This layer contains the Java files with that where accepted by their owner. It is used to create distributions of the software, giving an alternative to build a patched version also, including code that has not been accepted yet. The files are stored as normal java files of a NetBeans project.
- Temporary Files : Those are the copy made for every java file containing modifications which are still not approved by the owner of the code.
- XML Files : There is an XML for every Java file containing the information about the owner, permissions and information needed for the merging phase.
- The Emulated File System: Is the logical layer that manages the logical access to the physical files and presents the information about which part of the code is owned by which user and whether the local code has been approved or released by the owner. For this, it uses the information stored in the XML files. It also implements a transparent file system for the user merging the temporary files with the accepted ones when corresponds.

This file organization allows users to manage their owns versions of every file, but without loosing the real branch of the software being developed. The system should always have a copy of the “real files”, that is, the files containing code accepted by the owner. The reason for having a XML file for every file in the system is to simplify the merging phase every time a user has the chance to synchronize his working copy. The merging of the code, including the detailed and complex permission system of the system is almost impossible without any other information and very uncomfortable if this information is stored in the source itself. The information that the XML stores is as follows:

- name : The name of the entity.
- owner : user that owns this object.
- rev : the version of this object, incrementing in one for every modification made or accepted by the owner.
- modified : a boolean variable that allows to know if a certain object has been modified by a user that has not the rights.
- umod : user that modified the file without permission.
- uver : the version created by the user that is making the modifications without owning the object itself.

The field name has also some extra parameters:

- srcid : a description of the object that allows to distinguish for example two methods with the same name but different arguments.
- type : if the object is a variable, method, class, etc.

There exists some other parameters that could be optional like:

- extends : for classes that extends other ones.
- implements : for classes that implement others.
- overrides : for methods that override methods from a parent class.

5 Conclusions

With the system presented in this document it should be possible to support a real XP project development. Giving the chance to the development team to use a tool that has the flexibility enough to develop the software without having troubles because of a complicated tool. The simplicity behind this idea gives the real chance to give a competitive tool.

The rules that the system implements about ownership of the code for controlling the coordination of the participant's work also support this fact and add more flexibility, so that the user can create a project that works under the rules that are most similar to the way his/her team really works.

The fact of using an IDE that is widely known and used is very important, not only because there is no need to build one from scratch, but also because it does not represent a real adaptation to a new software for a development team.

Another important aspect about this work is the fact that many of the small projects developed in real life, such as small software for limited purposes, internal utilities for companies and also including many small Opensource projects are developed under the XP methodology under the conditions we described. This methodology is not completely supported by any software and this could be the chance to apply it completely supported by a development environment.

In order to implement the peer-to-peer communications among the online participants the system uses the JXTA™ [15] technology, which provides libraries and several APIs to make the implementation of peer-to-peer networks more reliable. Codebreaker use this technology to discover the participants of the developing team in the LAN and to establish a connection between them. JXTA also allows the system to be extended for many users, so that they can be connected from anywhere in the Internet, even trough firewalls.

CodeBreaker is still in the prototype implementation phase. To continue the work over this idea, we plan first to finish the development in order to test of the system in real environments. Starting with arranged experiments, with volunteers, and finishing with real teams, working on real projects. With all this information and the information of every tested team about it's past projects, the efficiency of this tool could be really measured and it could be possibly to conclude about it's effectiveness.

6 Acknowledges

This work was supported by a scholarship of the National Institute of Information and Communications Technology of Japan. We would also thank the support of Prof. Mitsuji Matsumoto of the GITI Institute of the University of Waseda, Tokyo, Japan.

7 References

[1] Schümmer,T. ,Schümmer,J. : Support for Distributed Teams in eXtreme Programming,In *eXtreme Programming Examined*, edited by Succi, Giancarlo, Marchesi, Michele , Addison Wesley, 2001.

[2] Bowen,S.,Maurer,F. : Designing a Distributed Software Development Support System Using a Peer-to-Peer Architecture, 26th International Computer Software and Applications Conference (COMPSAC 2002) , pp. 1087-1092, 2002.

[3] Berliner,B : CVS II:Parallelizing Software Development, 1989.

[4] SourceForge,http://www.vasoftware.com , last visited on 14 February 2006.

[5] Beck, K. : *Extreme Programming Explained*. Addison-Wesley,2000.

[6] Van der Hoek, A. , Heimbigner,D. , Wolf, A.L. : A generic, peer-to-peer repository for distributed configuration management, *icse*, pp. 308, 18th I nternational Conference on Software Engineering (ICSE'96), 1996.

[7] Magnusson,B. ,Asklund,U.,Minör,S. : Fine-grained revision control for collaborative software development, Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering, pp. 33 – 41, 1993.

[8] Xu1,B., Lian,W., Gao,Q. : A General Framework for Constructing Application Cooperating System in Wind, ACM SIGSOFT Software

Engineering Notes ,Volume 28 , Issue 2 (March 2003), pp. 15

[9] Neary,D. : Subversion - a better CVS, http://www.linux.ie/articles/subversion/ last visited on 13 February 2006

[10] Thomas,D. , Johnson,K. : Orwel, a configuration management system for team programming, Conference on Object Oriented Programming Systems Languages and Applications, pp. 135 – 141, 1988.

[11] Schümmer,T. ,Schümmer,J. : TUKAN: A Team Environment for Software Implementation. OOPSLA'99 Companion. OOPSLA '99, Denver, CO, pp. 35-36, 1999.

[12] IBM Rational ClearCase, Integrated SCM for Rational Developer products and Eclipse, ftp://ftp.software.ibm.com/software/rational/web/whitepapers/int-scm-rad-eclipse.pdf, White papers of IBM, December 2004

[13] Netbeans, Sun Microsystems, http://www.netbeans.org, last visited on 13 February 2006.

[14] Padberg,F. , Muller,M. M. : Analyzing the Cost and Benefit of Pair Programming, metrics, p. 166, Ninth International Software Metrics Symposium (METRICS'03), 2003.

[15] JXTA Technology: Creating Connected Communities , Sun Microsystems, http://www.jxta.org/docs/JXTA-Exec-Brief.pdf, last visited on 13 February 2006.