A.S. USTA METU 2006

TOOL SUPPORT FOR DISTRIBUTED AGILE SOFTWARE DEVELOPMENT

AHSEN SERKAN USTA

APRIL 2006

TOOL SUPPORT FOR DISTRIBUTED AGILE SOFTWARE DEVELOPMENT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHSEN SERKAN USTA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

APRIL 2006

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof.Dr. İsmet Erkmen
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Semih Bilgen
Supervisor

Examining Committee Members

Assist. Prof. Dr. Cüneyt Bazlamaçcı	(METU, EE)	
Dr. Özgür Barış Akan	(METU, EE)	
Dr. Şenan Ece Schmidt	(METU, EE)	
Dr. Altan Koçyiğit	(METU, IS)	
Prof. Dr. Semih Bilgen	(METU, EE)	

ABSTRACT

TOOL SUPPORT FOR DISTRIBUTED AGILE SOFTWARE DEVELOPMENT

Usta, Ahsen Serkan
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Semih Bilgen

April 2006, 103 pages

Agile Software Development has gained popularity with their people centric view and their common practices for developing software in today’s volatile business world where change on requirements is unavoidable. However; the efficiency of the project depends on the communication and the collaboration of the team, which are supported by the co-location of the team. But in some cases co-location of the team cannot be realized, thus agile processes should also support distributed teams. This point was observed by Kircher, Jain, Corsaro, and Levine [31] and they suggested Distributed eXtreme Programming (DXP) after they prepared a study using off-the-shelf software products in order to replace the effect of face-to-face communication on the efficiency of the application of agile processes with the aid gathered from tool support.

In this study some available tool support for distributed agile software development is investigated and a tool is developed and presented in order to support software configuration management as well as increasing collaboration and communication of the team. The tool is then evaluated from a user’s perspective and it is compared with some available software configuration management tools.

Keywords: Agile Software Development, Distributed eXtreme Programming, Software Configuration Management, Collaborative tool support

ÖZ

YAZILIM GELİŞTİRMEDE DAĞINIK ÇEVİK SÜREÇLER İÇİN ARAÇ DESTEĞİ

Usta, Ahsen Serkan
Yüksek Lisans, Elektrik – Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Semih Bilgen

Nisan 2006, 103 sayfa

Yazılım geliştirmede çevik süreçler, değişikliklerin kaçınılmaz olduğu günümüzün değişken iş dünyasında yazılım geliştirmek için sundukları ortak uygulamalar ve kişi merkezli bakış açıları ile hızla önem kazanmaktadır. Bu süreçlerle geliştirilen projelerin verimliliğinde takımın ortak bir alanda çalışmasının getirdiği iletişim ve işbirliğinin önemi büyüktür. Fakat bazı durumlarda takımın ortak bir alanda çalışması mümkün olmamaktadır; bu yüzden çevik süreçlerin dağınık takımları da desteklemesi sağlanmalıdır. Kircher, Jain, Corsaro ve Levine [31] bu noktanın farkına varmış ve hazır yazılımları kullanarak yüzyüze iletişimin getirdiği verimliliğin yazılımsal araç desteği ile sağlanabileceğini göstererek, çevik süreçleri kullanan Distributed eXtreme Programming (DXP) yöntemini ortaya koymuşlardır.

Bu çalışmada, dağınık çevik yazılım süreçleri için hazırlanmış olan araç desteği araştırılmış, yazılım konfigürasyon yönetimini desteklemek için bir araç hazırlanmış ve sunulmuştur. Bu aracın aynı zamanda dağınık takımlarda iletişim ve işbirliğini geliştirebilir bir araç olması önemsenmiştir. Bu araç kullanıcı açısından ve diğer yazılım konfigürasyon yönetimi araçları ile karşılaştırılarak değerlendirilmiştir.

Anahtar Kelimeler: Çevik Yazılım Geliştirme Süreçleri, Distributed eXtreme Programming, Yazılım Konfigürasyon Yönetimi, İşbirliği için araç desteği

To My Family

ACKNOWLEDGMENTS

I would like to express my deepest gratitude and appreciation to my supervisor Prof. Dr. Semih Bilgen who inspired, encouraged and supported me at all levels of this study.

I would like to thank to my friends Erkan Önder, who provided me a great working environment, Savaş Çetin, who motivated me during my depressed moments, and Başak Yılmaz, who presented me another kind of perspective for interception.

I would like to express my feelings of gratitude to my dearest Özlem Marul as supporting me at each step of this study and giving me inspiration during the endless work hours.

In addition, I would like to thank to my brother Serdar Usta for his motivating conversations and his support.

Finally, I would like thank to my family for their endless support and patience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASCM	Another Software Configuration Management
ASD	Adaptive Software Development
AM	Agile Modeling
CMM	Capability Maturity Model
COACH-IT	Component Oriented Agile Collaborative Handler of Integration and Testing
CSCW	Computer Supported Cooperative Work
CVS	Concurrent Versions System
DPP	Distributed Pair Programming
DSDM	Dynamic Systems Development Method
DXP	Distributed eXtreme Programming
FDD	Feature-Driven Development
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure Sockets
IDE	Integrated Development Environment
IEEE	The Institute of Electrical and Electronics Engineers
LD	Lean Development
MASE	MILOS Agile Software Engineering
MILOS	Minimally Invasive Long-term Organizational Support
MS	Microsoft
NFS	Network File System

NNTP	Network News Transfer Protocol
RCS	Revision Control System
SCM	Software Configuration Management
SMTP	Simple Mail Transfer Protocol
SSH	Secure Socket Shell
SVN	Subversion
TDD	Test-Driven Development
XP	eXtreme Programming
VCS	Version Control System
WebDAV	Web-based Distributed Authoring and Versioning

CHAPTER 1

INTRODUCTION

Software development processes began to take shape in order to improve productivity and quality of software in a repeatable and predictable fashion. They mainly aim to supply a consistent way to develop software, which would meet the expectations in terms of functionality, cost, or delivery schedule. Generally each process consists of similar steps, which are requirements analysis, specification, design and architecture, coding, testing, documentation, and maintenance. Their difference from each other depends on the execution style of these steps: i.e. all the steps can be executed in a big cycle at the end of which the project is completed (e.g. Waterfall process [32]) or these steps can be executed in small cycles in a continuous fashion and the project is decided to be completed (e.g. Incremental and Iterative processes [32]). Each of these steps are heavily documented in order to point out inputs and outputs of the step, and steps are executed one after each other within the scope of the documentations written and maintained during the execution of each step. The problem with those processes has been the cost of the changes that cannot be covered, and changes usually become unacceptable in later steps.

In today's volatile business world, change in the requirements is inevitable and so the lightweight methods, which are named as agile methods, are becoming popular as they have the ability to respond quickly to the changing requirements. Agile processes have a more people-centric view instead of process-centric view. Agile processes use feedback, instead of documentation and planning, as their primary control mechanism. The feedback is driven by regular tests and releases of the evolving software in small periods, also strong communication between team members increase the quality of the feedback. With the early feedback, changes can be controlled and included in the project at any time. Therefore, the expectations in terms of functionality, cost, and delivery schedule can be met at any level of the agile processes. Capability Maturity Model (CMM) describes good engineering and management practices and prescribes improvement priorities for software organizations [41]. Agile proponents dislike CMM, because they do not believe in the assumptions that CMM make for achieving its goals, especially predictability. However, Paulk [41] states that CMM and eXtreme Programming (XP) are complementary, while CMM, focusing on both the management issues to implement effective and efficient processes and on systematic process improvement, tells what to do, on the other hand XP, being a specific set of practices, tells how to do it. He also shows that XP addresses most of the level 2 and some of the level 3 practices, but not the level 4 or 5 practices of CMM.

XP is the most well known agile process as it is well documented and there exist many studies about the usage and application of XP [21]. It has four values: Communication, Feedback, Simplicity, and Courage. XP suggests the use of twelve practices, most of which are well-known software engineering practices stated and proved useful, during the project. With XP, successful software development is aimed even if there is uncertainty and rapidly changing requirements.

Agile processes can also support distributed teams, when collocation can not be realized, but in case of distributed teams the applicability of face-to-face communications decreases that weakens the efficiency of agile processes. Distributed eXtreme Programming (DXP) idea was suggested by Kircher, Jain, Corsaro, and Levine [31] after they prepared a study using off-the-shelf software products in order to replace the effect of face-to-face communication on the efficiency of the application of agile processes with the aid gathered from tool support.

There are basically four kinds of requirements to realize DXP that affect practices of XP: Communication, Coordination, Collaboration, and Infrastructure [31]. Communication between team members should be done in asynchronous or synchronous ways depending on the need of interaction. For example e-mail and discussion groups can be used for ordinary information exchange. Nevertheless, synchronous ways like videoconferencing should be used for daily meetings, and customer involvement. One important point is that the team should have known each other well to increase the efficiency of the distributed communication. Monthly meetings and regular activities can be planned to achieve that before or after iterations. Coordination systems help individuals to view, plan, and execute their own actions in relation to their co-workers' actions to their common goal. Collaboration creates a shared environment to enhance communication and coordination supports. Group decision support systems are used for collaborative problem solving in order to increase the quality of the decision or to reduce the time needed for it or both. Multi-user editors as well as desktop teleconferencing systems realized by an application sharing system can be used to enable a group of users to access a common document. To enable communication, coordination, and collaboration between teams and team members there should be software and hardware supports with common properties. For example, there should be some kind of connection to the Internet, application sharing systems, multimedia desktop or mobile computers, and tool support available to all team members.

The objective of this study is to contribute to the practice of DXP by proposing a new tool for its support. Specifically the aim is to develop a Software Configuration Management (SCM) tool, which enhances the management of projects and developers over web, and supporting collaboration and communication between developers using a client application over a client-server based networking architecture in order to increase the scalability while improving the collaboration.

In this study, the areas that necessitate tool support in case of DXP shall be investigated, and a tool profile shall be constructed in order to cover those areas. For this purpose, first, previous studies on the agile software development methodologies and their applications are investigated. As a next step, previous studies on tool support and existing tools for agile processes and their findings are investigated and examined. It is determined that there exist a need for tool support in the area of software configuration management, as the previous studies show that there exist enough tool coverage for other areas such as planning and management. Available open source tools in order to use for SCM are examined. The next step is to prepare a proposal for a tool that supplies the necessary coverage on the area of SCM according to the findings of the previous examinations. After the proposal is formed, the development of the tool is realized according to an IEEE 830 compliant format

Software Requirements Specification. Then the tool is demonstrated in a Software Engineering group project course (EE647) in order to gather feedback from the course audience. Later the tool is evaluated according to the feedback from EE647 course audience and the tool is compared with the similar tools. Finally, the proposed tool is analyzed using the findings of the evaluation. In addition, it is questioned that whether the usage of the tool covers the intended support for SCM during a DXP Project.

The proposed tool shall remove the dependency of the collocation of the group during the development process by means of communication and source access. The tool shall also increase the awareness of the team members about the operations of each other thus enabling information exchange by practical code reviews between pairs. The advantage of code reviews is that they may reduce development time and deliver more efficient and stable code that applies to established common standards. Distributed pair programming shall also be supported and encouraged, as it can ease adaptability of new members. The effectiveness of learning by practice can take place as pair programming and/or code review sessions with experienced developers decrease the time of adaptability. Therefore, integrating code review sessions with a Software Configuration Management Tool shall bring the team the opportunity of creating source code which is more readable, less erroneous, more understandable and adheres to common standards. As the code becomes self-explanatory, the necessity of documentation decreases, thus allowing developers and team to have more time for development.

This study does not aim to prepare a tool that covers all areas of SCM one by one as there are existing open source tools available for use with high maturity, for example in the low level areas of SCM such as Version Controlling. So existing open source tools are used and combined when it seems necessary in order to develop a lightweight tool. Especially, Subversion [49] will be used as the VCS tool and the tool can access and use Subversion repositories. The tool should have a general coverage of SCM and presents simple solutions that ease and speed up the development process in an agile way, which meets the need for tool support in the area of SCM in order to conduct a healthy DXP Project.

Chapter 2 of this study contains a survey on the Agile Software Development Methodologies and necessary tool support for them, especially for DXP. Agile processes, their limits and necessities, necessary tool support for agile processes, studies on tool support, agile SCM, and available open source tools for SCM are discussed in that chapter.

In Chapter 3, proposal of the Another Software Configuration Management Tool (ASCM) is developed and presented according to the findings of the literature survey.

Evaluation of the ASCM tool is presented and questioned in Chapter 4.

Chapter 5 presents conclusions regarding this study and suggestions for future work.

CHAPTER 2

LITERATURE SURVEY

In this chapter, first, a brief review of the literature on traditional and agile software development methods will be presented. Then, automated tool support provided for agile techniques will be examined. After investigating the availability of and requirements for tool support in the general context of distributed agile software development, lastly, software configuration management tools as available for and can be used in DXP shall be investigated and comparatively evaluated. This comparison will form the basis for the particular SCM tool developed within the scope of this thesis study and described in chapter 3.

Agile methods emerged as an alternative to the traditional software development methods, to keep up with the increasing speed of technological changes and business needs of today's industrial world. Because traditional methods necessitate a complete set of well-defined requirements should be documented and elicited in order to begin the implementation of the process, which are hard to be fulfilled clearly and completely because of the rapid changes in requirements. In order to respond to the inevitable changes experienced, several methods and practices have been developed by several consultants [14]. Besides those methods and practices, tools are developed in order to decrease the burden of the traditional methods and ease their application, so people can concentrate on the development of the system, instead of the process.

Development of tools mainly focuses on the improvement of the process by proposing solutions on several areas. Tools are mainly used for

- the establishment of plans and control their execution,

- to edit and check documents,
- to communicate and to exchange results.

Tools come out with many useful solutions and find themselves many application areas, but as the coverage and usage areas of tools increase, their complexity increases and their learning curve steepens, too. Thus, tools also bring the burden of their process of training and adaptation, besides the bureaucracy of the traditional methods. Also tool support is usually limited, in that it does not improve the quality of the process, but only eases the application of the process.

As the development world changed, practitioners realized that traditional methods did not work as they expected, and they found out that people-oriented and flexible practices having generative rules should be developed in order to handle the changing requirements better.

In January, 2001 the Agile Software Development Manifesto emerged [3], when seventeen of the agile proponents came together to discuss the new software development methods. The four statements of the Agile Manifesto are:

- Individuals and interaction over process and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

The first statement emphasizes people-oriented values, e.g. premium people and face-to-face communication. The second statement emphasizes working code as the primary artifact, in contrast to traditional methods with document-driven development. The third statement states that the on-site customer is a necessity of today's software development world. The fourth statement emphasizes embracing change, as change is inevitable.

The agile methods do not differ from traditional methods so much with the practices they apply, but their focus and the values behind them are different. In contrast with traditional methods, agile methods actually do not depend on the efficiency that tool support may bring. However, they can benefit from the tool support in many areas, so as the way their name suggests agility; they also lightened the use of tools [30].

2.1 Agile Software Development Methodologies

Traditional methods are often called as heavyweight methods, as their bureaucratical structure slows down the development. Agile, which has meanings like readiness of motion, nimbleness, activity, etc., is used for naming the new methodologies, formerly named as lightweight methods in contrast to traditional methods' slow, and plan-driven processes. According to Highsmith et al., being agile means being able to "Deliver quickly.", "Change quickly.", and "Change often." [23]. Therefore, agile methods have common characteristics such as iterative development, a focus on interaction and communication, and reducing the size of documentation. In addition, they suggest different practices in order to achieve their goals. There exist several methodologies classified as agile, the well known ones are Adaptive Software Development (ASD) [63], Agile modeling (AM) [64], Crystal family [65], Dynamic Systems Development Method (DSDM) [66], eXtreme Programming (XP) [21], Feature-Driven Development (FDD) [67], Lean Development (LD) [68], and Scrum [47].

Being one of the most popular agile methodologies, eXtreme Programming (XP) [21] is producing more and more empirical studies that help XP to stay in the focus of researchers and practitioners who are interested in agile methods. And also it is a fact that number of these studies are increasing day by day, and the results from which are discussed in conferences, and eWorkshops. According to the results from some of these conferences and eWorkshops the needs, limits and capabilities of Agile Methods can summarized as:

- Size seems to be the most important factor that may directly affect communication, which is very important for Agile Methods. Increase in the project size needs an increase in the team size, and/or in the number of teams, which causes a significant decrease in the effectiveness of face-to-face communication. Therefore; scale-up strategies, for example teams of teams, are needed. The scalability issue is addressed in an eWorkshop and its results are summarized in an article [43].

- Experience is important for an agile project to succeed, but experience with actually building systems is much more important than experience with Agile Methods. In the 2nd eWorkshop [50] on Agile Methods at Fraunhofer Center for Experimental Software Engineering it was estimated that 25%-33% of the project personnel must be competent and experienced , but the necessary percentage might even be as low as 10% if the teams practice pair programming due to the fact that they mentor each other.
- Formal training requires less time than that of traditional methods because face-to-face communication and pair programming help minimizing what is needed in terms of training, because people mentor each other.
- Reliable and safety-critical projects can be conducted using Agile Methods. Performance requirements must be made explicit early, and proper levels of testing must be planned. It is easier to address critical issues using Agile Methods since the customer gives requirements, sets explicit priorities early and provides continual input. And also test-driven development that XP uses controls the development phase strictly. However, Boehm suggests using plan-driven methods if the change in requirements is less than %1 [8], which is mostly the case of reliable and safety-critical projects, while strict regulations must be applied for them.
- The three most important success factors are culture, people, and communication. Agile Methods need cultural support, i.e. the organization has to support the viewpoints of Agile Methods, and otherwise they will not succeed. Competent team members are crucial. Agile Methods use fewer, but more competent people. Rapid communication is supported by physically co-located teams and pair programming. Close interaction with the customer and frequent customer feedback are critical success factors.
- Early warning signs can be spotted in agile projects, e.g., low morale expressed during the daily meeting. Other signs are production of useless documentation and delays of planned iterations [33].

- Refactoring should be done frequently and of reasonably sized code, keeping the scope down and local. Large-scale refactoring is not a problem, and is more feasible using Agile Methods. Also refactoring is inexpensive with Agile Methods, in contrast to that with plan-driven methods [8].
- Documentation should be assigned a cost and its extent be determined by the customer. Many organizations demand more than that is needed. The goal should be to communicate effectively and documentation should be the last option.
- Distributed teams seem to be supported by Agile Methods, but the case that they decrease the applicability of face-to-face communications. Therefore, it is needed to examine and find solutions for using Agile Methods with distributed teams. A pioneering work on this issue is Distributed eXtreme Programming (DXP) by Kircher, Jain, Corsaro, and Levine [31].

In XP there are twelve practices followed during the course of basic activities and covering the fundamental principles, which are planning game, small releases, metaphor, simple design, tests, refactoring, pair programming, continuous integration, collective ownership, on-site customer, 40-hour weeks, open workspace, coding standards. These practices by themselves are not effective, but together they are generally accepted to create a coherent method. In addition, most of these practices are common sense practices that disciplined processes use. XP emphasizes continual redesign that decreases the need for detailed design documentation. Pair programming results in improvement in problem solving and decrease in code failure [48]. Testing early prepares developers to the implementation phase with a better understanding of the requirements.

2.2 Tool Support for Agile Methods

Tools can be used and developed in order to cover the needs and to extend the limitations of Agile Methods. Generally, agility refers to tools that are flexible and easy to use, and they have to respond to user needs or can be configured in that manner easily. Also it should be easy to make changes, while Agile Methods are also adaptive to changes. Additional services are needed in case of distribution for supporting communication, coordination, and cooperation. Tool support can be viewed from five perspectives, which are communication, coordination, and cooperation; planning; development and documentation; configuration management; process automation [30]. As Kelter, et al. [30] classified and detailed these perspectives, the important and notable points from those five perspectives can be stated as follows:

2.2.1 Communication, Coordination, and Cooperation

1. Communication should be supplied in synchronous and asynchronous ways in order not to lose the efficiency of the agile methods [31].
2. There should be application-sharing system in order to access to documents. Also change notifications, and access rights should be sent or given to the relevant team members.
3. Schedules, appointments, and deadlines should be announced to plan and manage the project.
4. Whiteboard and multi-user editors should be used to improve communication.
5. Electronic representation of physical objects should be realized to increase awareness [46].

2.2.2 Planning

1. Tools integrated with the development environment should be used by everyone, because according to agile methods everyone is responsible for planning his/her own tasks and estimating his/her own efforts.
2. Change notifications about relevant data should be seen when accessed.
3. Tools for monitoring and control should also be used to compare actual efforts and dates with the plan. Therefore; the manager can keep an eye on the project and also can immediately react to deviations.
4. Tools should be easy to use and simply should offer the needed functions.

2.2.3 Development and Documentation

1. There should be test tools like Junit [28], and they should be automated.

2. Accessibility of diagrams and tools should be controlled and managed from a central repository, in order not to lose the version management.
3. Multi-user editor support should exist for editing and viewing documents and diagrams.
4. Tools should be easy to use and can be tailored or configured according to the users' need.
5. Refactoring tools should be integrated in order to detect structures to be refactored and to help performing refactoring.

2.2.4 Configuration Management

1. There should be only one development path and one executable version of the system existing at any time.
2. Access rights should exist.
3. Simultaneous access to files should be synchronized and users should be warned and directed to work as a pair if appropriate [46].

2.2.5 Process Automation

1. Configuration management activities, e.g. freezing the code after iteration, or change notifications, should be automated.
2. System generation should occur in an automated way.
3. Module tests should be executed after each integration, and notification should be sent to relevant team members about the test results.

In the field there exist mostly coordination, planning and tracking tools, which are built upon the web technologies while being easily accessible is a key value. These tools generally take eXtreme Programming features as a basis, such as User Stories, Release Planning, and Iterations, and they usually give metrics such as velocity (as in XP), and burn down charts (as in Scrum [47]).

Development is usually supported with test tools like Junit, as they ease test-driven development. As simplicity being one of the principles of Agile Software Development Methods, refactoring has an important role in realizing that principle. There exist some IDE's (Integrated Development Environment) that have some features supporting refactoring, like Eclipse [20], IntelliJ IDEA [27], Borland JBuilder [9].

Configuration management is usually realized using generally CVS (Concurrent Versions System) [17] or its open-source equivalent Subversion [49].

Process automation is very common in Agile Software Development Methods, as test suites should be repeated over and over again, and as continuous integration being one of the key practices.

As Agile Methods suggest small teams and they necessitate the collocation of the teams, they are mostly restricted with collocation and the advantages that collocation brings. At this point Distributed eXtreme Programming (DXP) get into the scene with solutions for how to distribute teams without losing the efficiency that eXtreme Programming asserts.

2.3 Distributed eXtreme Programming

Distributed eXtreme Programming (DXP) was first proposed by Kircher, et al. [31] in order to use eXtreme Programming practices in a distributed team, thus reducing constraints and cost while increasing mobility and customer involvement. They suggested that only four of the twelve XP practices, which are planning game, pair programming, continuous integration, and on-site customer, require co-located team.

The factor that makes XP work, as mentioned in Section 2.1, above, is the effective and strong communication between team members, customer, and the manager. To achieve that level of communication, the team has to be physically collocated, and the customer should be available on site all the time. But in some cases collocation might not be possible or might not be wanted because of several reasons like physical distribution of development teams according to the organization’s structure, individual constraints, large team sizes, new models of work like e-lancing works or virtual software development, etc. There came the question by Kircher, et al. if it is possible to use XP without loosing its effectiveness in such cases where co-location does not exist [31]. And they have used the term “Distributed eXtreme Programming (DXP)”, which is an extension to XP, for the first time at Proceedings XP in 2001.

Remedies should be found for the practices of XP, which require the existence of co-location to realize DXP successfully. With an investigation of XP practices, it can be seen that planning game, pair programming, continuous integration, and onsite customer, only four of the twelve practices, require a collocated team. Also refactoring and testing can be effectively done in case of collocation, but as they depend usually on pair programming, a remedy for pair programming should be applicable to them as well [48].

In case of planning game; the story cards, which are the main artifacts of communication to provide information about the project to the customers and the developers, are used to inform developers and customers about the current status of the project. The way they are created, modified, re-estimated, sorted or selected depends on communication. Therefore, in DXP the story cards have to be remotely accessible, and the team members should be able to communicate with each other for effective use of the story cards.

In case of pair programming two programmers sitting in front of the same machine work on the same part of code. While one of them writes code, the other investigates the code, thinks about the design, makes suggestions and comments the code. The roles change often, i.e. about every 15 minutes, and the formation of the pairs change with every new task. So communication and application sharing is important for pair programming. In case of Distributed Pair Programming (DPP) [48], the programmers should hear each other and both must be able to see the same code on their individual screens while only one of them can change the code at any time.

In case of continuous integration, after a task is completed it should be integrated to the system. Integration takes place several times a day, and with the collective code ownership practice, they both affect the whole system. Therefore, programmers should be informed of changes and integrations.

In case of on-site customer, the customer is a part of the team and he/she is available all the time to answer questions of programmers and to write acceptance tests. So that there should be some kind of communication like videoconferencing or desktop teleconferencing between the team and the customer.

There are also other issues to deal with like metrics and awareness. The former is related with the state and the progress of the project, programmers are informed of those by the state chart in case of XP in which the team is physically co-located in a subconscious way. The latter is related how the presences and the conversations of the team members effect each member in the manner of knowledge exchange and how knowledge spreads between the team. In DXP the former can be supplied using a similar way that XP uses, but the latter is hard to realize like the way physical co-location realizes.

The remedies offer creating a virtually shared environment with the assistance of several tools and today's evolving Web technologies. Different approaches were proposed by several researchers: Kircher, et al. [31] extend the assumptions of XP with connectivity between team members, e.g. Internet, information exchange with e-mail, configuration management to enable collective ownership, application sharing for pair programming, videoconferencing for planning game and customer involvement, and familiarity between team members; Maurer [35] investigated open source projects from the view of virtual teams' communication, collaboration and coordination styles over Internet technologies; Schümmer and Schümmer [46] applied results from the research area of Computer Supported Cooperative Work (CSCW) to the XP methodology. Virtual teaming and Distributed Pair Programming are investigated by Laurie Williams, extensively [48].

Generally, the solutions for XP practices are proposed as video conferencing and application sharing software support for planning game, remote pair programming by sharing Integrated Development Environment(IDE), continuous integration on the machine at the team site or on the machine that the development was done, on-site customer by using video conferencing.

2.4 Tool Support for Distributed eXtreme Programming (DXP)

Kircher et al. [31] developed a tool called Web-Desktop [31] to support DXP by using off-the-shelf products like MS Netmeeting [40], CVS [17], CUSeeMe [18]. They suggested using a combination of synchronous and asynchronous communication, and generating a way to serialize change access when working on common code for reducing the possible conflicts when using the integration tool.

Maurer et al.[35, 36, 37] came up with the idea of analyzing open source projects which are usually realized with virtual software teams in order to get the answers for the following questions:

- How the communication, collaboration and coordination of virtual software teams realized?
- How the information and knowledge are shared?
- What tool support is used?
- At what points shortcomings occur?

The results of their analysis shows that open source projects maintain Web site(s) as a base for information exchange, Internet-based configuration management systems (mostly CVS) is used for accessing source code and documentation, web-based issue-tracking systems are used to record bugs and for managing workflow in order to fix them, discussion groups and mailing lists are used for proposing new features and extensions. As seen most of the information is available for developers in pull-mode, which means developers usually get the information by accessing the project site or CVS whenever they want and/or have time. Push-mode is used for distributing bug reports, for discussing new features, or for on-line support in e-mail format [35].

Maurer et al. [35, 36, and 37] suggest improving the work process of virtual software teams using DXP and open source processes as a baseline. While XP projects are more coordinated than open source projects, project coordination should be done in some way that tasks should be assigned to developers, deadlines should be set, and overview on the current state of the project can be get by XP team, whereas team members can access their to-do lists and retrieve information relevant to their work easily. Synchronous communication is necessary in order to provide face-to-face communication in a manner to realize planning game and pair programming where necessary. Active notification and information routing should be provided in order to exchange knowledge continuously. Process execution should be integrated with knowledge management by keeping the contents of experience base up-to-date and integrated with everyday processes in order to make organizational learning possible and to keep up better with stuff changes.

In agile methods knowledge sharing is much more efficient than in traditional methods, while agile methods suggest face-to-face communication instead of document oriented information exchange. Moreover, in agile processes there exist several practices encouraging knowledge sharing, which are release and iteration planning, pair programming and pair rotation, on-site customers in case of XP, daily Scrum meeting, cross-functional teams, and project retrospectives in Scrum [12]. In case of release and iteration planning knowledge about system requirements and domain is shared between team and customers. In addition, discussions refine the requirements to a better set and they enhance understanding. During pair programming sessions, tacit knowledge is shared between developers, and pair rotation enables knowledge exchange among the team. In daily scrum meetings, team members share their progress and goals with each other and exchange information about similar problems they ran into. Cross-functional teams reduce the development time by enabling the experience sharing between different members of the team. Project retrospectives facilitate the identification of success factors and problems during the project thus enabling continuous learning. Therefore, tool support for agile methods has to realize knowledge sharing somehow in the above manner.

There exist various tools supporting distributed development in several perspectives and they all can be used for supporting DXP in different ways. Most of these tools are open-source but there are some commercial products as well. There are process support tools such as MILOS (Minimally Invasive Long-term Organizational Support) [37], MASE (MILOS Agile Software Engineering) [38], COACH-IT (Component Oriented Agile Collaborative Handler of Integration and Testing) [42]; development support tools such as TUKAN [48], Sangam [45]; management support tools such as xPlanner [54], XPWeb [55]. In the following subsection, each of them will be presented briefly.

2.4.1 Some Available DXP Tools

<u>MILOS:</u> [37]

MILOS supports agile software development with the use of some XP practices. User stories can be created during planning game and modified during the development life cycle. System maintains a development schedule of releases, iterations, user stories, and tasks. Coordination and initiation of pair programming is also provided by using the MS NetMeeting integration. MILOS also includes a metrics utility that produces size and complexity metrics for packages, classes, and methods.

<u>MASE:</u> [12, 38]

MASE is the extended version of MILOS with several new features for supporting distributed XP, which are

- User stories: A new product type for user stories is added, and whenever a new user story is added MASE automatically adds a task for implementing this story into the task list.
- Release and iteration planning: MASE allows easily defining and changing releases, iterations, user stories and tasks. And also it provides awareness on what is going on in the project.
- MS NetMeeting integration: Distributed Pair Programming and synchronous communication is provided via MS NetMeeting [40].

Team members can easily access their workspaces using a web browser by connecting and logging on to the MASE server. The list of current projects, user stories, currently available tasks, task estimation and pair programming facilities are available from their workspaces.

MASE eases knowledge sharing with the aid of Wiki technology [52]. MASE makes providing information nearly as easy as accessing information. With Wiki users can access, browse, create, structure and update any web pages using only a web-browser. And moreover Wiki technology comes over maintenance problems by automatically creating links from a wiki page to particular topics pages if the names of those topics are mentioned in that page. Also MASE provides full-text searching capabilities on any content.

COACH-IT: [42]

Scalability is an issue of agile methods, but it can be overcome as in the case of Schwaber's "Scrum of Scrums" by using an architecture-centric approach. An architecture-centric strategy depends upon up-front planning, which is not suggested by agile methods, in order to define which teams will work on which modules and how to integrate those modules at the end. However, up-front planning can be done in an agile way just by focusing on what is required. Continuous integration should also be done not only for each module itself but also for all modules in order to justify their dependencies and relationships. Also tests should be written by module users not module providers, in order to enhance the understanding over inputs and outputs.

As a summary, an architecture-centric software can be combined with agile software development processes retaining the spirit of agile by following these guidelines: [42]

1. Design the module architecture in a quick and lightweight way
2. Provide the architecture in a format that is flexible to change
3. Require test first design at the interface level
4. Tests are written by module users not module providers
5. Test authors act as customers for dependant modules(in addition to real on-site customers)
6. Module teams define their own product backlog of user stories
7. Do continuous integration of the module based system

COACH-IT executes the following sequence: [42]

1. Users define an architecture using the COACH-IT input web application
2. Multiple repositories are monitored for code changes in each module
3. When a change is detected the module and related modules are downloaded

4. The modules are deployed and tests are run to ensure interface compatibility
5. Teams are notified directly of any problems via electronic mail
6. The "health" of the system is available to the teams via a web page

Using the COACH-IT tools a set of modules with jUnit [28] tests assigned to their interfaces can be defined quickly by developers. Module names and(optionally) descriptions/annotations, module repository locations, module file locations, module interfaces, module team contact information (e-mail), module relationships (unidirectional), relationship test associations, test repository locations, test file locations and test contact information (e-mail) are defined within the Architecture Definition language file, which is the core of COACH-IT [42]. Only these few items are required as user input to create a simple architecture for continuous integration. CruiseControl [16] continuous integration tool is used for monitoring modules, and ANT build file is called when a modification is detected on a module in order to perform the integration and testing. Individual developers and teams are also directly notified by using electronic mail in case of a test failure or a change in system health.

TUKAN: [48]

TUKAN is a synchronous distributed team-programming environment, which applies some groupware research results to the XP domain in order to solve the problems when XP is carried out by distributed teams. Groupware is defined as "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment." [48] Communication, coordination, and collaboration are three categories of groupware issues relevant to XP. Communication can be realized in an asynchronous way (as in e-mail) or in a synchronous way (as in video-conferencing). Coordination systems are necessary in order to plan, view and execute actions of developers, and they contain workflow management systems, document management systems, and version management systems. In collaboration, basic communication support and coordination support are combined in a shared environment. Besides groupware categories TUKAN supports awareness of the workspace by using some color coding depending on the virtual distance.

SANGAM: [45]

Sangam is an extension for Eclipse IDE [20], which is developed for distributed pair programming. It enables sharing of the Eclipse platform between a client-server pair and enabling the exchange of code writing on the same editor.

XPWeb: [55]

XPWeb is a web-based management tool for XP projects. It supports planning game: user can define user stories, decompose them into tasks, assign members to tasks and evaluate them. Then he/she creates iterations and chooses user stories and tasks for iterations. It improves feedback by continuously updating progress of implementation and comparing it with automatically calculated theoretical progress. It enables centralizing of XP resources, such as metaphor, CVS [17], tests and other documents.

XPWeb is very powerful as it integrates planning, process tracking, documentation, testing and continuous integration altogether in one platform. It fully follows the XP practices, and eases the distributed development in that manner. It is also well supported with help topics at each feature and their usages.

xPlanner: [54]

XPlanner is a project planning and tracking tool for eXtreme Programming (XP) teams. Xplanner was developed to support planning process of XP.

Xplanner enables simple planning model; virtual note cards; support for recording and tracking projects, iterations, user stories, and tasks; smart continuation of unfinished stories (unfinished tasks copied, copied stories are crosslinked); distributed integration token (with email notification); online time tracking and time sheet generation at individual/team level; metrics generation (team velocity, individual hours, etc.); charts for iteration velocity; ability to attach notes to stories and tasks (with attachments); iteration estimate accuracy view; page showing task and story status for individual developers and customers.

XPlanner supplies some kind of communication improvements with the use of virtual note cards, notifications, and attaching notes to user stories. Also it shows metrics at different levels, which may be useful for detecting and solving possible problems earlier.

Table 1. Relevant tool support for selected XP practices [30]

XP Practices	**Tool Support**
Planning game, on-site customer	CSCW (Computer Supported Cooperative Work) systems, groupware; document and application sharing, distributed management of schedules and appointments
Small releases, tests, continuous integration, collective ownership	Distributed Software Configuration Management (SCM) systems; fast building processes; automated and fast testing; management of test cases, results, and feedback; report generation
Refactoring, simple design	Support for refactoring source code and models; handling of partial design information; analysis and assessment of current state of the design; executable design
Pair programming, open workspace	Multi-user editors; teleconferencing; document and application sharing; virtual representation of physical objects

Kelter et al. [30] summarized the relevant tool support for XP practices in Table 1 according to the requirements of tools listed above for agile methods. They mention that tool support for many tasks already exists but there are many requirements, which need to be addressed. They conclude that an agile software development environment should be easy to use and it should offer enough flexibility as well as offering additional services for supporting collaboration and coordination, for automation and for exchanging information.

2.4.2 Evaluation of Tools

Tools are evaluated according to the key issues in agile software development methods, scalability, as agile teams usually refer to 8-12 people. and distribution, as agile teams are supposed to be collocated. And also key values of agile software development methods, which are communication, collaboration, and coordination, are taken as a basis. Some of XP practices, which are needed to be supported in case of distribution, planning game, pair programming, continuous integration, on-site customer as Kircher, et al. stated [31], are also taken as evaluation criteria. One of the reasons for taking XP practices as evaluation criteria is most of the available tools are built upon XP practices and principles.

Distribution is the main and perhaps the easiest issue to overcome, as web technologies being came to such a state, in which accessibility is not an issue anymore. Moreover, communication support of web technologies make the necessity of building web-based tools in order to decrease and perhaps remove the necessity of collocated teams.

In order to overcome distribution problems, and as agile methods' strength mostly depends on people-oriented behaviors, tools should offer remedies for communication, collaboration, and coordination issues that could arise inside teams. Most of the available tools are designed to support collaboration and coordination, and using these two they aim to support communication generally (XPWeb [55], COACH-IT [42]). Some of them are also gives communication support with the aid of additional tools (MILOS [37], MASE [38], they both use the MS NetMeeting integration). Xplanner uses distributed integration tokens (with email notification) for informing team members. TUKAN was equipped with basic audio and chat support, and also it has awareness support, which informs team members working on the same part of code, or working on the code parts which could be affected by each other. As being an easy way of asynchronous communication some tools use Twiki [52] support in order to maintain and manage topics, and knowledge sharing between team members (MASE [38], Xplanner [54]).

As most of the tools are built upon the principles and practices of XP, they have to find some solutions to overcome the XP practices, which need collocated teams, such as planning game, pair programming, continuous integration and on-site customer. Most of the presented tools are built for project planning and tracking, therefore; planning game is realized in similar ways. First user stories are added to the database, project manager selects team members, and after making a release plan and user stories are added to the iterations, user stories are associated with the team members, and also team members can be paired for some user stories implementation. xPlanner has the feature of virtual note cards and attaching notes on user stories, which eases the planning game phase. Pair programming is supported with MS NetMeeting integration in MILOS and MASE, but TUKAN supports pair programming by itself supplying an environment for development. Also SANGAM is just built for realizing pair programming over Eclipse platform. Continuous integration is supported by most of the tools by use of CVS repositories. On-site customer is not a problem while all of the tools are web-based, and customer can be reached by using web technologies at any time.

Scalability is an issue questioning whether Agile Software Development Methods could be successful with large teams and in large projects. COACH-IT is the only tool, which tries to overcome this problem using a centralized-architecture, from which teams could be communicated and coordinated.

Table 2 summarizes the comparison of some available DXP tools:

Table 2. Comparison of some available DXP tools

Tool / Supports	MILOS	MASE	COACH-IT	TUKAN	Sangam	XPWeb	xPlanner
Planning Game	2	2	2	2	0	2	2
Pair Programming	0	1	0	3	3	0	0
Continuous Integration	1	1	3	1	0	3	2
On-site Customer	1	1	1	2	0	2	3
Scalability	0	0	2	0	0	2	2
Distribution	2	3	2	3	2	3	3
Communication	1	3	3	3	2	2	3
Coordination	3	3	3	3	0	3	3
Collaboration	1	1	2	3	3	2	3
Extras	MS NetMeeting Integration	MS NetMeeting Integration	CruiseControl, jUnit, ANT integration	Awareness support	N/A	Integration with CVS; Documentation Support included	Twiki support

0: Does not Support 1: Supports with the aid of additional tools 2: Supports 3: Fully supports

2.5 Requirements for Distributed XP Tools

- Tool should be web-based, but does not necessitate high bandwidth except for certain development issues, like pair programming.
- Tool should include a web-based chat or some kind of bulletin board, in order to communicate among members during user story creation, planning game, and release planning.
- Tool should be user friendly, easy to learn and support enough documentation about its features which is easy to access (XPWeb is the well-documented and user friendly tool among the others)
- Tool should be integrated with some kind of VCS, which gives directly accessible links to project iterations inside the tool.
- Tool should be able to send notifications about changes to the appropriate team members implicitly.
- Some kind of process automation and reporting should be realized by the tool for test suites and daily builds.
- Tool should keep some metrics, and track the project using them. And also there should be some kind of history about metrics, like MASE which keeps metrics about the past projects and use them as an experience base for the current project.
- As test-driven development should be realized, tools like Junit [28] should be integrated with the tool. And tool should be integrated with an IDE which supports refactoring during the development phase.(For example, Eclipse [20])
- Some kind of whiteboard or virtual note cards in xPlanner [54], could be useful during the realization of the first phase of XP.
- During pair programming if there exist some kind of audio communication supported by the tool, this would increase understandability and efficiency of communication without affecting the concentration of the developers.
- Some kind of awareness could be supplied by the tool in order not to pre-arranging pairs but by notifying each developer working on similar tasks.

Some kind of history should be kept by the tool about the specifications of the tasks and there should be some kind bulletin board or knowledge sharing platform both are equipped with some kind of search engine in order to aid developers and enhance the communication between team members.

As there exist a comprehensive tool support for planning and coordination functions for agile methods, investigating the software configuration management area, which is a major necessity and also may support the practices such as collective code ownership, pair programming, refactoring, and code review would make more sense. So in the next part available tools that specifically address Software Configuration Management (SCM) in the context of agile software development are examined.

2.6 Software Configuration Management (SCM) Tools

Generally speaking, SCM tools are usually defined as Version Control Systems. After agile methodologies emerged with some best practices like daily builds, pair programming, and collective code ownership, SCM tools shall also include some features in order to automate and control code changes and integrations as well as collaboration over code, such as enabling code reviews and refactoring.

2.6.1 Common Features

SCM tools are usually defined as Version Control Systems (VCS), which have different shapes and sizes, but their common aim is to keep revisions of source code using a common code repository for any project. Some systems support Atomic Commits, which means that the state of the entire repository changes all at once. Thus, keeping the history of the system, one can access to an older version easily.

Most common VCSs allow merging of changes between branches. This means that changes committed to one branch will be committed to the trunk or another branch as well, with one automatic or at least semi-automatic operation. There are different ways of merging, like three-way merge, picking a common ancestor, diffing the new and the old versions against that, attempting to combine the diff into one, and merging if they can and complaining to the user if they can not.

A distributed VCS allows the cloning of a remote repository, producing an exact copy. It also allows changes to propagate from one repository to another. In non-distributed VCSs, a developer needs repository access in order to commit changes to the repository. That leaves developers without repository access as second-class citizens. With a distributed VCS each developer can clone the master repository and work on it, later propagating his changes to the master repository.

Another common factor is whether the repository allows versioned file and directory renames (and possibly copies well), i.e. whether it is possible to copy, move or rename files and directories without loosing the tracking of the VCS.

In the subsection, some available SCM tools that can be used in DXP are examined.

2.6.2 Some Available SCM Tools

Aegis: [2]

Aegis enforces a development process requiring that change sets "work" before integration into the project baseline. While CVS provides a repository; aegis provides a repository, a baseline, mandatory reviews and mandatory testing. Aegis may be configured to use almost any history tool (such as Revision Control System (RCS)) and almost any dependency maintenance tool (such as make), although traditional make may not be sufficiently capable. It is not networked, and all operations are done via UNIX file-system operations. As such, it also uses the UNIX permissions system to determine who has permission to perform what operation. Despite the fact that Aegis is not networked, it is still distributed in the sense that repositories can be cloned and changes can be propagated from one repository to the other. Allowing network access requires using a file system such as Network File System (NFS).

Being an SCM system, Aegis tries to assure the correctness of the code that was checked in. It:

- Manages automated tests, prevents check-ins that do not pass the previous tests, and requires developers to add new tests.

- Manages reviews of code. Check-ins must pass the review of a reviewer to get into the main line of development.

Arch: [6]

Arch is an advanced SCM specification with multiple, independent but compatible implementations. Arch is widely considered even more ambitious than Subversion, mainly on account of its support for fully decentralised (as opposed to merely distributed) repositories: In Arch, any branch or developer's private work area can be treated as a repository of its own, with a global name space for developers, repositories, and branches.

Arch is a distributed version control system. It does not require a special service in order to set up a network-accessible repository, and any remote file-service service (such as File Transfer Protocol (FTP), Secure FTP (SFTP), or Web-based Authoring and Versioning (WebDAV)) is a suitable Arch service.

Arch supports versioned renames of files and directories, as well as intelligent merging that can detect if a file has been renamed and applies the changes cleanly.

CVS: [17]

CVS (Concurrent Versioning System) is extends RCS by bringing the network and concurrency support in order to control concurrent editing of sources by several users working on releases built from a hierarchical set of directories.

Despite its popularity, CVS has its limitations. For example, it does not support file and directory renaming. Furthermore, binary files are not handled very well. CVS is not distributed and the commits are not atomic. Also CVS makes branching and merging difficult. There are some available CVS extensions, which are aimed to improve the CVS usage.

DARCS: [19]

DARCS is CVS-replacement SCM, handling all metadata, supporting fully decentralised repositories and advanced branching / patch-handling. In DARCS, each copy of the source is a fully functional branch and DARCS patches have a set of properties, which make possible manipulations that could not be done in other revision control systems.

Monotone: [39]

Monotone is a distributed (networked) SCM with a flat peer model, cryptographic version naming and metadata certificates, decentralized authority, and overlapping branches. It works out of a transactional version database stored in a regular file. Network communication is mediated via HTTP, netnews (NNTP), or SMTP transport.

Monotone supports renames and copies of files and directories. It has a command set that aims to be as CVS-compatible as possible, with some necessary deviations due to its different philosophy. Monotone is a distributed version control tool. It can help automate many tedious and error-prone tasks in group software development.

- Transmit changes to files between colleagues.
- Merge changes made by colleagues.
- Make notes about opinions of the quality of versions of files.

Subversion: [49]

Subversion is specifically designed as a CVS replacement, fixing all of the long-known design flaws in CVS: It adds versioning of directories, file renames, and handling of all file metadata including symbolic links. Revision numbering is per-commit instead of per-file. Commits are atomic. Designed-in network operations are over either WebDAV, or a lightweight custom protocol called svnserve, which runs on port 3690 and can be anonymous or authenticated by a svnserve-specific password file. Alternatively, svnserve can provide authentication through SSH.

Subversion aims to create a better replacement for CVS. It retains most of the conventions of working with CVS, including a large part of the command set, so CVS users will quickly feel at home. Aside from that Subversion offers many useful improvements over CVS: copies and renames of files and directories, truly atomic commits, efficient handling of binary files, and the ability to be networked over HTTP (and HTTPS). Subversion also has a native Win32 client and server

Superversion: [51]

Superversion is a multi-user distributed version control system based on change sets. It aims to be an industrial-strength, open source alternative to commercial solutions that is equally easy to use and similarly powerful. In fact, intuitive and efficient usability has been one of the top priorities in Superversion. Superversion is based on changes and change sets. A change can be:

- Creation, deletion or modification of an individual file
- Modification of the list of files to ignore

2.6.3 Comparative Evaluation of SCM Tools for XP

- Aegis creates a difference from other SCM tools because of its testing and build tool integration, which is also an important point for Agile Methodologies. However, because of its poor network support, it does not support distributed organization of development teams.
- Arch, with its large compatibility of networking services, can ease the networking and also is very popular. As it is fully decentralized, this would cause problems in iterative development with iterations around two weeks, such as developers may lose the track, and also integration of software becomes an issue from a developer point of view.
- CVS is the most known SCM tool and has a lot of variants and utilities.
- Darcs is the only SCM tool, which supports operations on patches, and patch tracking.
- Monotone is fully decentralized and uses SHA encryption for security.
- Subversion supports file and directory renaming, copying, and moving.

- Superversion has an integrated GUI that eases the use and eases viewing and tracking the changes. Also it has a good help documentation.

All of these SCM tools can be used for an Agile Software Development Project, but Aegis with its continuous integration and testing supports would be much useful than the others. Nevertheless, a little burden will also appear for the development team as in addition to unit tests, someone has to prepare and maintain full path tests for testing integration before and after each build. Also the major restriction of Aegis is the team should use the same network in order to commit and check-out, which limits the distribution of the team.

Fully decentralized SCM tools should be thought of when there exists a geographical distribution of developers. Most of those SCM tools support peer-to-peer networking and exchanging differences using clones of the master repository. But the restriction would be the short iteration periods for eXtreme Programming (XP), and the time zone differences between developers may cause the project go out of track.

As a result, for Agile Methodologies instead of fully decentralized SCM tools, centralized ones with web access would be better and the management and maintenance would ease. As it can be seen these tools are for code versioning and they are sufficient and mature for versioning and keeping repositories. So instead of a fully functional version control tool, enhancing the capabilities of code versioning system clients in order to improve collaboration between individual developers and/or team will decrease the necessity of collocation and the effect of the team size on agile processes efficiency. An integrated SCM tool using one of these tools as code versioning system and extending its capabilities with management of developers and projects as well as improving the communication and collaboration between developers would have some improvements on development process.

Scalability of teams is also another issue on agile processes, and it has unsustainable effects on the information exchange in case of distributed teams. There exist instant messaging tools, but usually an account is needed and the account should be exchanged with other developers working on the project which should be done manually and which may not be suitable as the developers do not know each other in case of distributed teams. Also only messaging between developers may not solve the problem of information exchange thus increasing collaboration, so a tool supporting the ability of reviewing and editing codes as a group or as pairs of developers without annoying distributed teams with details of the realization of collaboration is needed.

Table 3 summarizes the comparison of some available SCM tools that can be used in DXP.

Table 3. Comparison of some available SCM Tools

Tool / Supports	Aegis	Arch	CVS	Darcs	Monotone	Subversion	SuperVersion
Atomic Commits	2	2	0	2	2	2	2
Changesets	3	3	0	3	3	2	2
Merging Quality*	2	3	2	2	2	3	3
Branching	3	2	2	2	2	2	3
Directory and File Name/Rename Versioning	2	2	0	2	2	3	2
Distribution	2	3	0	2	2	1	3
Networking	0	3	2	2	2	2	2
Web Interface	2	2	1	2	0	1	0
Collaboration support	0	0	0	0	0	0	0
Testing and Build	1	0	0	0	0	0	0
Extras	Uses UNIX basic tools	Uses any remote file service	Easy to use	Patch support	Supports Encryption	Copy/move and rename operations allowed	Integrated GUI, Easy to deploy and use

0: Does not Support 1: Supports with the aid of additional tools 2: Supports 3: Fully supports

* 0: N/A 1: Average 2: Good 3: Superior

CHAPTER 3

ANOTHER SOFTWARE CONFIGURATION MANAGEMENT TOOL (ASCM) PROPOSAL

The requirements for the tool developed within the scope of this thesis study have been derived based on the survey of literature presented in the preceding chapter. These requirements have been presented in IEEE 830 compliant format in Appendix A. A brief review of these requirements follows:

3.1 Specifications of ASCM

SCM is usually interpreted as source code management; therefore, in the area of Version Control Systems (VCS) there exist a lot of research and projects for supporting that research. After examining the open-source Software Configuration Management Tools in Section 2.6, it is observed that most of the available work has been done on Code Versioning Systems, and in that area, there exist generally available mature tools. However, nearly all clients of these tools are independent applications and they are not supporting collaboration of teams. So a client supporting collaboration using automated notifications of the events and let the developers get acquainted with the significant problems earlier, without needing an extra effort or configuration would be useful for distributed teams in order to manage an efficient team work as in case of a collocated team. Also in distributed software development, information exchange depends on the comments inlined with the code, which may not be sufficient in some cases. Thus, the quality of the written code decreases with the increasing number of distributed developers working on the code. Reviewing and refactoring code by pairs or groups have major effects on the quality, as well as the information exchange between developers having different skills. Therefore; enabling code reviewing and code editing and refactoring using online chat sessions among a group of developers would improve the information exchange as well as the quality of the written code. At this point, the exact need for ASCM becomes more understandable: a SCM tool, which enhances the management of projects and developers over web, and supporting collaboration and communication between developers using a client application over a client-server based networking architecture in order to increase scalability while improving collaboration.

As the agile software development methodologies bring the “people factor” in front of every process, the tool should supply an interface covering the access and control of versioning control system, as well as keeping the data about the projects, and developers and easing their access to the resources. Also the update and alert properties of such tools should be in an advanced state, such that in case of a conflict, or the possibility of a conflict occurrence related parties should be informed about the situation as well as necessary action, which prevents any commits, should be taken. By the same way VCS should inform developers about the changes, e.g. when a commit operation is performed, both on the local repositories and on the base repository when possible, without waiting the developer to update changes manually.

By taking those as a basis, ASCM should provide support in the following manner:

- Supply management support by:
    - Providing a central web based project and developer repository;
    - Enabling the creation of the project source code versioning repositories automatically;
    - Enabling the access rights of developers when they are assigned to a project.
- Supply a Version Control System Client which supports
    - Standard code versioning system operations, such as commit, update, merge, diff and resolve conflict operations;
    - Messaging between developers of project;
    - Code reviewing and editing during online chat sessions;
    - Mailing notifications to developers when they are inaccessible.
- Supply a Version Control System Server which

    - Checks local changes on working copies and inform developers working on same files which may cause conflicts;
    - Enables the communication between clients by supplying them necessary information.

For this Software Configuration Management System Subversion [49] is used as Versioning Control System, presented in Section 2.4, and the Client API for Subversion by TmateSoft [62] is used for client application's Subversion based operations.

3.2 Structure of ASCM

ASCM mainly consists of three parts:

- Portal,
- ASCM Server Application,
- ASCM Client Application.

The functionality of those parts can be summarized as follows:

- Management Portal
    - Configurations of the tool users and management of access rights,
    - Repository management,
    - Project management,
    - Developer management.
- ASCM Server Application
    - Communication between client agents,
    - Used for checking local changes and informing related parties,
    - Collecting information over clients and generating necessary actions,
    - Used for handling Subversion base/client errors,
    - Uses notifications in order to inform related parties when necessary.
- ASCM Client Application
    - JavaSVN API Library [62] is used for Subversion integration.
    - Used for the access of the Subversion base/client repositories.
    - Uses Notifications in order to inform related parties when necessary.

- Used for performing standard Subversion operations
- Generates automatic notifications when a change observed.
- Enables developers to review and edit codes and then commit the changes after online chat sessions.
- Enables messaging between developers.

3.3 Detailed Functionality

3.3.1 Management Portal

System administrator can create projects over web management portal. Each time a project is created a repository will also be created with the same name of the project.

Developers can sign up to the system over web management portal with the necessary information belonging to them (i.e., username, password, name, e-mail address, information about the capabilities of the developer). System administrator can approve/disapprove the registrations of developers thus accepting them to access the system. Developers are informed about their registration status via mail.

The records of not approved registrations are automatically deleted by the system after a specified period.

After signing up to the system developers can select projects and apply for working on them.

System administrator can assign developers to the projects they applied according to their capabilities thus enabling the developers to access the repository of the project and enabling them to communicate with other developers of the project. Each developer can work on only one project at any time period. After a developer is assigned to a project the necessary information is sent to developer via mail.

3.3.2 ASCM Server Application

Keeps the client information and status (i.e., connected or not) on project basis, and exchange those information with the other connected clients of the same Project.

Collects information from clients about the working copies and checks whether two users are working on the same file. In case of such a situation, it notifies clients to start a code review session.

When a commit operation is on progress, warns other clients in order not to permit any commit operations during that period. Moreover, notifies clients after commit operation succeeded, so that the clients ask users for performing update operation.

Sends notification mails to offline users about the changes on files that they are the last authors of the files, i.e. the last revision of the files are committed by those users.

3.3.3 ASCM Client Application

It is the end user interface that the user performs all repository operations as well as performs messaging with other developers of the project.

Developer can perform standard repository operations such as Checkout, Import, Commit, Update, Status, Diff, Add, Tag, History.

Developer can start a chat session in two ways:

- By opening a source file to start a code review and editing session with selected developer(s).
- By opening a chat session in order to exchange information with selected developer(s).

Client notifies user in case of a successful commit operation, so that the user can update his/her working copy and continue working with latest codes.

Client prevents user to perform a commit operation while another user is performing a commit operation.

3.4 Usage Scenarios

In the subsections, 3.4.1 and 3.4.2 below, step-by-step scenarios for typical usage will be given for different functionalities of the tool.

3.4.1 Management Portal Usage Scenarios

3.4.1.1 Logging on to the system

- User connects to the main page URL of the Web Management Portal using a Web Browser.
- Selects the user type as Admin or Developer
- Enters Username and Password.
- If the entered Username and Password are correct User is directed to the related page according to user type (i.e., Administrator Page or Developer Page).

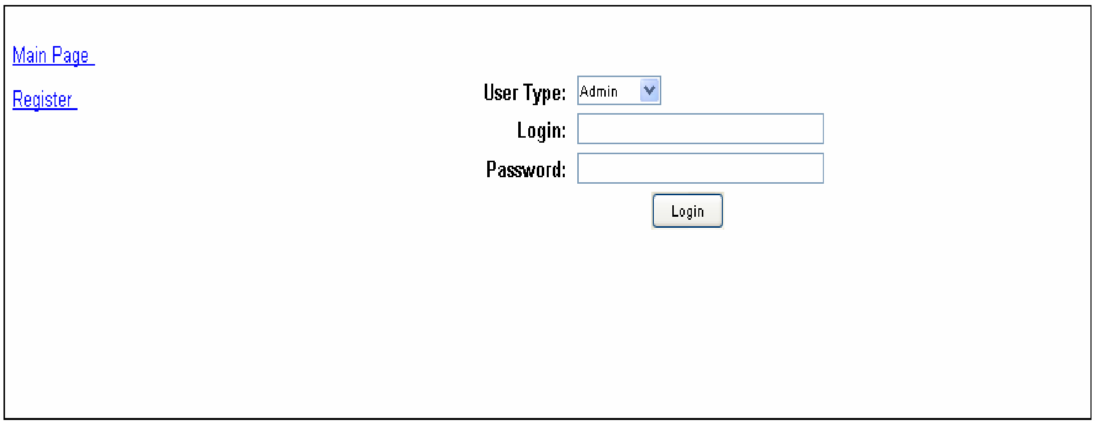


Figure 1 Login Screen

3.4.1.2 Creating a Project

- Administrator logs on to the system.
- Selects the Create Project link.
- Fills up the Project related information.
- Clicks on the Create Project button.

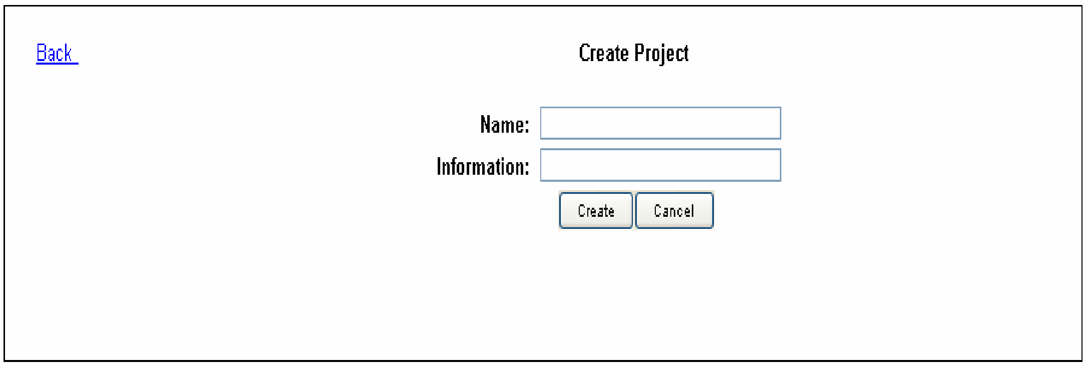


Figure 2 Create Project

3.4.1.3 Registration of a New Developer

- User connects to the main page URL of the Web Management Portal using a Web Browser.
- Selects the Register link.
- Fills up the Developer related information.
- Clicks on the Register button.
- A scheduled mail is sent to Administrator including the list of new registered developers automatically, if there exists new registrations during the period.

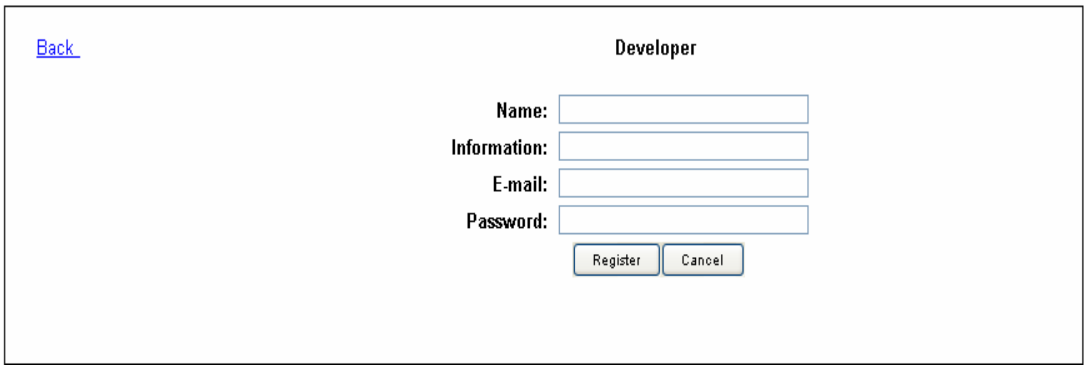
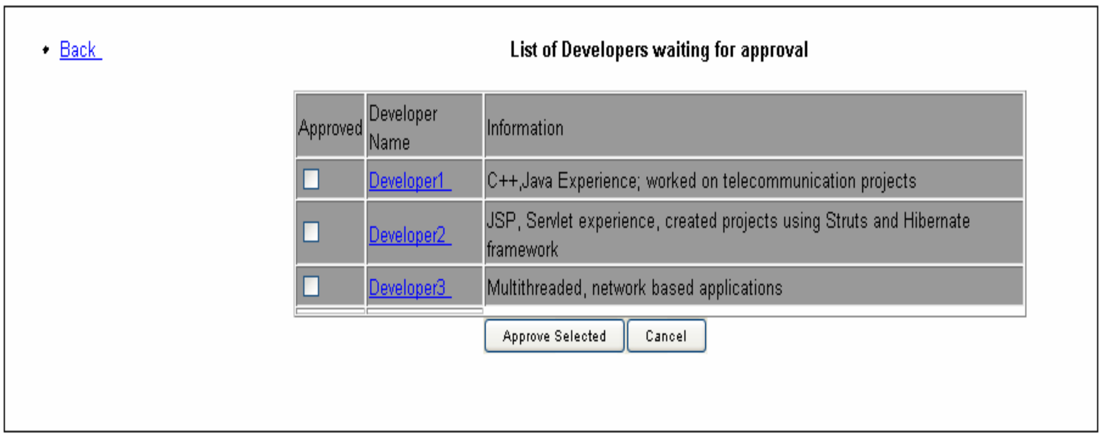


Figure 3 Registration of Developer Screen

3.4.1.4 Approval of Developers

- Administrator logs on to the system.
- Selects the Developer Approval link.
- Overviews the Developer information. Selects Developers to approve.
- Mail notifications are sent to developers about their approval status. If they are approved they are invited to select projects.



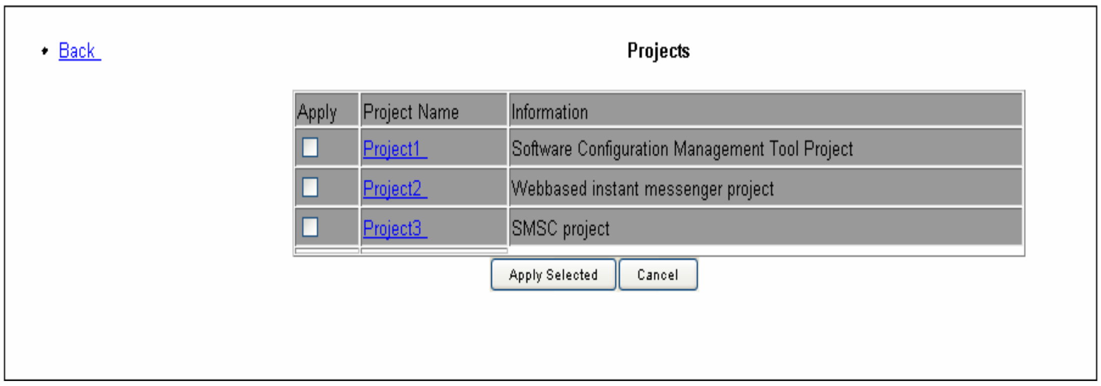
List of Developers waiting for approval

Approved	Developer Name	Information
☐	Developer1	C++,Java Experience; worked on telecommunication projects
☐	Developer2	JSP, Servlet experience, created projects using Struts and Hibernate framework
☐	Developer3	Multithreaded, network based applications

Figure 4 Developer Approval Screen

3.4.1.5 Developer applies to Projects

- Developer logs on to the system.
- Selects the Apply to Projects link.
- Projects are listed in a table including the information of the project, number of developers.
- Developer selects from the list of existing Projects by selecting checkout boxes.
- Clicks the Apply button.



Projects

Apply	Project Name	Information
☐	Project1	Software Configuration Management Tool Project
☐	Project2	Webbased instant messenger project
☐	Project3	SMSC project

Figure 5 Apply to Projects Screen

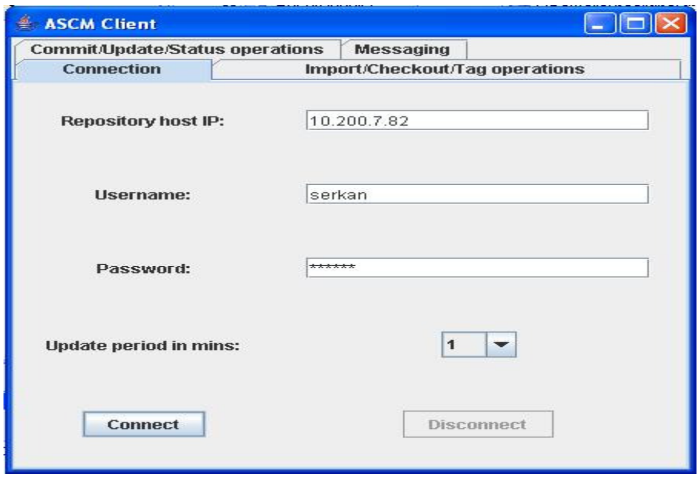
3.4.1.6 Administrator Assigns Developer to a Project

- Administrator logs on to the system.
- Selects the Assgin Developer to Project link.
- Unassigned developers are listed in a table with the name of the projects they applied for.
- Administrator selects a project for each Developer.
- Clicks the Assign button.

3.4.2 ASCM Client Usage Scenarios

3.4.2.1 Developer connects to the system

- Developer starts Client Application.
- Enters the server IP that is sent via mail.
- Enters username and password.
- Selects an update period.
- Clicks Connect button.
- Server authenticates the user and sends back the project name that the user is assigned.


ASCM Client
Commit/Update/Status operations
Messaging
Connection
Import/Checkout/Tag operations
Connect
Disconnect

Repository host IP: 10.200.7.82

Username: serkan

Password: ******

Update period in mins: 1

Figure 6 Connection Screen

3.4.2.2 Importing source codes to the repository

- When the project is first created, a source code base, which has the project directory structure base and some base source code files, should be imported to the repository.
- Developer clicks the Import button and selects the locations of source code base to be imported.
- The result of the Import operation is viewed in a dialog box.

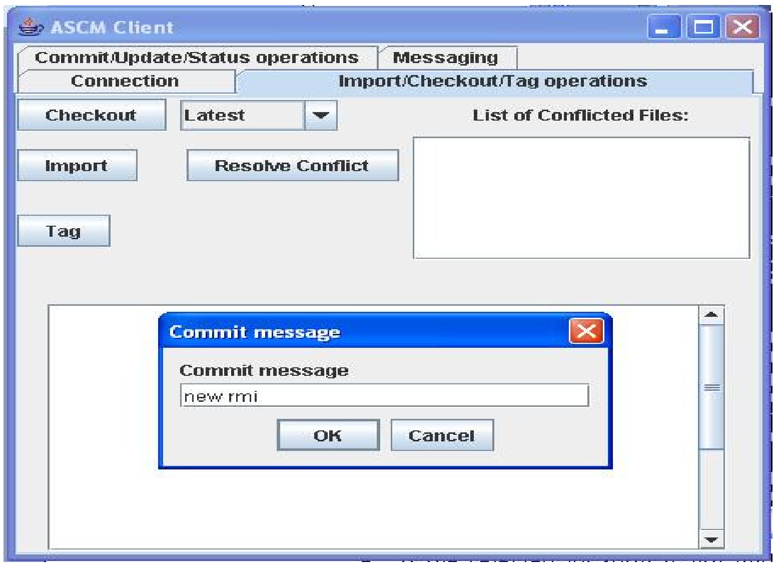


Figure 7 Import/Checkout/Tag Operations Screen

3.4.2.3 Initial check out of source codes to a Working Copy

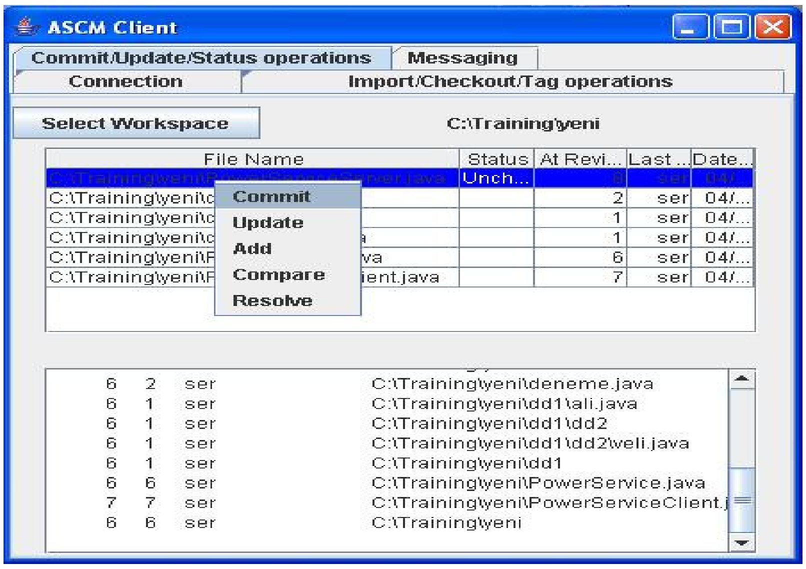
- Developer selects a local working copy location at the Repository Pane.
- If the selected location is not under version control, user is notified about the situation and warned to perform a checkout operation.
- Developer selects from the options of checking out the Latest Revision of the Project, checking out specified revisions of files that will be selected, or checking out files by tag.
- According to the selected option, user is either warned to select files to checkout from repository tree, or the whole project is checked out to the

specified location by given tag or latest revision depending on the choice of the developer.

- The result of the Checkout operation is viewed in a dialog box.

3.4.2.4 Update working copy

- When commit operations by other users performed to the repository user is notified about changes and asked to perform update operation.
- After the client connects to the system, if there exist any change on the repository user is asked to perform update operation.



C:\Training\yeni

File Name	Status	At Revi...	Last ...	Date...
[illegible]	Unch...	6	ser	04/...
C:\Training\yeni\c		2	ser	04/...
C:\Training\yeni\c		1	ser	04/...
C:\Training\yeni\c a		1	ser	04/...
C:\Training\yeni\F va		6	ser	04/...
C:\Training\yeni\F ient.java		7	ser	04/...

```
6   2   ser          C:\Training\yeni\deneme.java
6   1   ser          C:\Training\yeni\dd1\ali.java
6   1   ser          C:\Training\yeni\dd1\dd2
6   1   ser          C:\Training\yeni\dd1\dd2\veli.java
6   1   ser          C:\Training\yeni\dd1
6   6   ser          C:\Training\yeni\PowerService.java
7   7   ser          C:\Training\yeni\PowerServiceClient.j
6   6   ser          C:\Training\yeni
```

Figure 8 Repository Operations Screen

3.4.2.5 Automatic Status change notifications

- Client checks the status of the working copy on the selected period.
- Client automatically shows the status changes of the files on the Repository Pane.

3.4.2.6 Committing files to repository

- Modified files are shown in the Repository Pane. User can select files to commit.

- First the diff window showing the changes are viewed. And the user is asked to review the changes and continue with the commit operation. User can cancel the operation at this step.
- If user wants to continue with the operation he/she is asked to enter comment for this commit operation.
- The result of the Checkout operation is viewed in a dialog box.

3.4.2.7 Showing history of files

- User can select a file to view the files history by right clicking on the file and selecting Show History link from the Repository Pane.
- History of the selected file is viewed in a window as a list.

3.4.2.8 Comparing two revisions of a file

- User can select a file to view the diff of a file by right clicking on the file and selecting Compare link from the Repository Pane.
- User is asked for which revisions to be compared. Default comparison is made between the latest revision and the modified file. However, user can select two previous revisions to be compared.
- Comparison of two revisions is viewed in a window with marks to show differences.

3.4.2.9 Starting a chat session

- User can select another user or users to start a chat session. User is also asked for the topic of the session.
- Selected users are notified about the chat request and asked for acceptance.
- If users accept the request a chat session is started between users.

3.4.2.10 Starting a code review session

- User can select another user to start a code review session.
- User is asked for the topic of the session and the source file to be reviewed.

- Selected users are notified about the code review request and asked for acceptance.
- If users accept the request a code review session is started between users.
- In code review session, users can make modifications on the source file and the user who started the session can save the file and commit the file.

3.4.2.11 Warning user after a commit operation

- When a user performs a successful commit operation, client notifies the server about the operation and server notifies the other clients.
- Clients notifies users about the commit operation, the files committed and asked to perform update operation.

3.4.2.12 Warning users in case of a conflict possibility

- As server collects information about the local changes of users, when server identifies that users make modifications on same file(s) server notifies the clients about this situation. Clients warn users and suggests starting a code review session.

CHAPTER 4

EVALUATION OF ASCM

In this chapter, first the usage areas and the evaluation criteria of ASCM described in Chapter 3 shall be discussed. Then, the feedback taken from the EE647 course in the METU EEE Department [56] audience after the presentation of the tool is discussed and the improvements on the tool based on the feedback described. Lastly, the comparison of the tool with available Software Configuration Management Tools and Clients shall be presented and the advantage of the ASCM as well as limitations shall be discussed.

4.1 Usage areas

DXP was discussed in detail in Chapter 2, and the factors limiting the efficiency are determined as the factors mainly effecting communication, collaboration, and cooperation. ASCM is developed in order to be used for distributed software development especially in the case of DXP where the communication between team members is of great importance as well as the teamwork being essential for the success of the project. ASCM supports the team in many ways like the application of coding standards, information exchange between team members, enabling the adaptability of new developers, pair programming and group development encouragement, scalability of the team, resistance to developer changes. Thus ASCM shall find itself a place in such cases where size of the team is variable, collocation of the team during development activities is not possible, frequent change of developers leads to information loss and communication problems, and adaptation of new members becomes time loss for other developers.

4.2 Evaluation Criteria

Software evaluations can be conducted as taking the merits and limitations of software as a basis and comparing these with similar software from a user's perspective. The motivation behind the software and its capabilities as well as its functionality, usability, efficiency and complexity should be the key factors of evaluation. In addition, gathering feedback from different user profiles increases the effectiveness of the evaluation.

In this study, the evaluation of ASCM depends mainly on the user's perspective, thus the gathered results are subjective to users' opinions. As the underlying technologies of Subversion clients are similar, the reason for using user's perspective for evaluation becomes much clearer. The data transfer duration depends mainly on the connection type and the protocol used for accessing the repository. As all the clients interact with the repository using Subversion supported protocols, the performance of the clients depends on the connection type used. However, from the user's perspective the usability and the flexibility of the clients vary, so the evaluation is conducted mainly taking these features as basis.

The features and limitations of ASCM should be evaluated in a perspective where distributed team support is a key usage area, thus:

- Information should be kept and delivered in an efficient way
- Communication between team members should be flexible and effective.
- Teamwork should be encouraged.
- Adaptation of new members to the team should not create a burden.
- Information loss should be prevented in case of frequent developer change.
- Team members shall be aware of each other.
- Code conflict possibilities should be decreased.

4.2.1 The Motivation

ASCM is mainly developed for distributed group usage where the agile processes, especially XP, are chosen as software development process. ASCM removes the dependency of the collocation of the group at the development process by means of communication and source access as well as supporting information exchange by practical code reviews between pairs and also supports and encourages distributed pair programming. ASCM not only supports the standard versioning system commands, but also automates largely used practices thus bringing flexible control while decreasing the complexity of usage. ASCM also increases the awareness of users as checking the files on which users are working and the operations they are performing on the repository, thus decreasing the possibility of conflicts and time losses. Code reviewing sessions increase the quality of the code by both increasing the readability and decreasing the possibility of errors, and also the code reviewing sessions ease the exchange of information by means of time and understandability.

Code reviews may reduce development time and deliver more efficient and stable code that applies to established common standards. Code reviews allow teams to catch and correct more errors not only syntactical but also logical errors in the development cycle. They are an essential part of teamwork where teamwork is defined to be essential to successful projects as agile methodologies XP demonstrate. Code reviews improve and encourage communication between team, while helping developers to avoid common mistakes as well as giving the chance to learn from each other while practicing. Perhaps the most important advantage is that the end product (i.e. source code) is easily maintainable even if team members change, because code reviews ensure that the information is distributed amongst team members.

Code reviews should also be included in the Software Configuration Management as they are related with the quality of the source code. So integrating code review sessions with a Software Configuration Management Tool shall bring the team the opportunity of creating source code which is more readable, less erroneous, more understandable and adheres to common standards. As the code becomes self-explanatory the necessity of documentation decreases, thus allowing developers and team to have more time for development.

It is also important to include the new members to the team and so to the development as quick as possible. At this point, the effectiveness of learning by practice can take place as pair programming and/or code review sessions with experienced developers decrease the time of adaptability. Information about the project, the standards and the work has been done can be transferred to the new member during code review sessions.

4.2.2 The Capabilities and Functionality

ASCM enhances the standards of Software Configuration Management Tools by adding the synchronous/asynchronous communication features thus creating a network between team members in order to inform each other about the activities of others. Also the members are encouraged and reminded to use the communication features of this network. The functionality of the ASCM is described in detail in Chapter 3. In summary ASCM:

- can be used for project management by means of working developers and repository management;
- is a Software Configuration Management Tool client enabling access to Subversion code versioning system repositories;
- supports distributed teams;
- encourages team members to communicate with each other;
- encourages code review sessions;
- presents the developers an awareness that they are notified by each others operations performed on repository.

Distributed development is usually preferred when collocation of the team is not possible, which in turn brings communication problems between team members. The necessity of collocation can be removed during implementation phase by ASCM, which conducts effective communication between team members, in order to increase the scalability.

4.2.3 Usability and Complexity

ASCM has a user-friendly self-explanatory interface, which is easy to use. The tool can be used on cross-platforms, so the developers can work on their preferred OS. The tool has automated many sequential functions enabling the users to perform an operation in a few steps. The notification mechanism helps developers to be aware of each change on the repository thus enabling them to work with the up to date repository. Online editing and reviewing sessions as well as chat sessions between team members improve the efficiency of the development process thus encouraging developers to gather and exchange information during these sessions.

4.2.4 Feedback and Improvements

Version 1 of the tool was presented in EE647 Course in March 2006, with a demonstration of its capabilities of automated notifications and code versioning system operations. In the demonstration, briefly, a project and a developer is created using the web management portal, then using the same portal developer is assigned to the project thus the access permissions to the repository of the project is given to the user. Then the user connected to the system using ASCM Client and created a local workspace area. Tool checked the selected location whether the location is under version control or not. If the location is not under version control, user is asked to perform a checkout operation with a dialog window. User can approve the checkout operation or decide to perform the operation manually. Thus, the tool warns the user about this necessity. After the checkout operation is performed, user can work on local workspace, editing the source files or creating new source files. Client has a task scheduled to check the changes between local workspace and repository. If a change is detected user is asked to perform an update operation or commit operation depending on whether the change is on the repository side (i.e., a user performs a commit operation) or the change is on the local workspace side (i.e., the user changed a source file). Also the user can decide to automate these operations thus the client automatically updates itself when a change on the repository is detected or tries to perform a commit operation when a local change is detected. It is important the note that even if the commit operation is described as automated, it necessitates the approval of the user, so that user decides to perform a commit operation or not. Also the tool notifies the user about conflicts and commit operations. Version 1 of the tool lacked some features like browsing the repository and performing operations in order to view revisions and performing compare operations.

The feedback gathered from EE647 course audience can be summarized as follows:

- The notification mechanism of the tool was found useful.
- The automation of commit and update operations were found to be risky as errors can lead to broken codes.
- The lack of browsing repository was found as a serious handicap, which may cause users to perform operations without confidence.
- As far as the tool kept revision history of source files the unavailability of returning to a previous revision or the unavailability of viewing and comparing revisions were found as a degrading factor on tool usability.

In general, version 1 of the tool was found to be useful but not compatible to be used in real life projects, so based on this feedback some improvements on the available features were implemented as well as new features in order to increase the usability and feasibility on distributed usage are added and/or implemented. The second version of the tool consists of the following features:

- Repository browsing is now fully supported, which enables performing operations on selections.
- File comparison based on revisions can be performed.
- Users are notified when they are working on same files with other users which may cause conflicts.
- Users can communicate using chat sessions.
- Users can conduct code reviewing and editing sessions.
- Users can learn on which files the other users are working.

With the improvements and additional features, Version 2 of ASCM supports distributed teams with software configuration management practices and enhances the communication and information exchange between team members. Thus increasing the quality of end product as well as minimizing the know-how and information loss during the project.

On the negative side, ASCM covers only the standard repository operations as they are used on a regular basis, and it does not support functions for the use of administration of a Subversion repository fully.

4.2.5 Comparison with other SCM tool clients

Some available SCM tools were presented in detail in Chapter 2. However, tool clients were not specifically examined in that comparison as each client should support the features of its SCM tool. In this part the available clients for Subversion repositories regarding to their features are described and a comparison between those clients and ASCM tool client is given. The comparison is made between JSVN, eSvn, SmartSVN, SubCommander, TortoiseSVN and ASCM.

4.2.5.1 Some Available SCM Tool Clients

JSVN [58]

JSVN is a Java based Swing GUI Subversion client. It mainly has the following features:

- Ability to connect to a remote Subversion repository
- Simple abstraction of Subversion commands
- Checkout, status, log, info, update, diff, commit Subversion commands
- Authentication to a Subversion repository

It requires that the Subversion client should be installed on local machine.

eSvn [57]

eSvn is a GUI front-end to the Subversion revision system. It mainly has the following features:

- Usage of the Subversion commands like: Checkout, Import, Export, Update, Commit, Add, Delete, Copy, Move, Merge, Switch, Revert, Log, Blame
- Enabling creation of profiles for the usage of different working directory and workspaces
- Enables browsing of a SVN repository
- Performs Quick Diff, External Diff and 3-way Diff
- Shows changed items
- Shows status of files/directories in real time

Subcommander [60]

It is another GUI front-end for Subversion repository. It mainly has the following features:

- integration of repository browser and working copy status view
- colored working copy status view to quickly see which files/folder have changed.
- easy branch and tag handling.
- has drag and drop support to move or copy files and folders.
- easy handling of multiple working copies (e.g. different branches) from the same project.
- Usage of the Subversion commands like: Checkout, Import, Export, Update, Commit, Add, Delete, Copy, Move, Merge, Switch, Revert, Log, Blame

TortoiseSVN [61]

TortoiseSVN integrates seamlessly into the Windows shell (i.e. the explorer), and is developed as an extension for the Windows Explorer. It mainly has the following features:

- It shows the status of every versioned file and folder by small overlay icons.
- Easy access to Subversion commands
- All Subversion commands are available from the explorer context menu.
- It enables the browsing of the repository

SmartSVN [59]

SmartSVN is an innovative multi-platform client for Subversion, the designated successor of CVS. SmartSVN has powerful features like built-in File Compare/Merge, Change Report or Tag and Branch handling. It mainly has the following features:

- Built-In Repository Browser to check out, create directories, remove and copy files and directories, log and annotate files

- Usage of the Subversion commands like: Checkout, Import, Export, Update, Commit, Add, Delete, Copy, Move, Merge, Switch, Revert, Log, Blame
- File Table Filtering (show/hide ignored, unversioned or unmodified files)
- Built-In File Compare with detection of inline-changes and the possibility to edit files
- Individual File Table sorting, also by multiple columns simultaneously

4.2.5.2 Comparison

The clients can be compared according to the features they present to the end user. The most important difference between the ASCM Tool and the other clients is ASCM tool supports distributed teams and presents a communication channel over which code review sessions and information exchange can be realized. ASCM tool supports standard Subversion commands as well as the other clients, but SmartSVN [59] has superiority over other clients in this area. Except SmartSVN [59], other clients does not enable advanced merging and branching management, however SmartSVN [59] presents these capabilities in its professional version which is not free. All of the clients present user friendly and easy to use interfaces, thus increasing the usability and learnability. SmartSVN [59] presents editors for code editing, reviewing, and comparing. However, no collaboration support exists on any of the clients.

All of the clients show the status of the files in local workspace in real time, and in colored views. TortoiseSVN [61], SmartSVN [59], Subcommander [60] and eSVN [57] support the Subversion commands fully, while ASCM and JSVN [58] only support the commonly used VCS commands of SVN. SmartSVN [59] also presents real-time repository indexing with the use of which users can follow the file change histories and logs on the repository. This feature of SmartSVN [59] is similar to the ASCM’s notification mechanism, but ASCM’s notification mechanism has also conflict preventative functionality and also it encourages users to work together by supplying users with the awareness of the others.

Table 4 presents a summary of the comparison of ASCM with some other Subversion clients.

Table 4. Comparison of ASCM with some other Subversion clients.

Tool / Supports	**JSVN**	**eSVN**	**Subcommander**	**TortoiseSVN**	**SmartSVN**	**ASCM**
Subversion Command support	2	2	3	3	3	1
File Status View	1	2	2	2	2	2
Branching	1	2	2	2	3	1
Diff	1	3	2	2	3	1
Collaboration Support	0	0	0	0	0	3
Notification of Repository Changes	0	0	0	0	1	3

0: N/A 1: Average 2: Good 3: Superior

CHAPTER 5

CONCLUSION

In this work, four major tasks have been accomplished. First, agile software development methodologies have been examined from the following perspectives: the motivation behind their appearance when compared to existing methodologies like Waterfall, incremental and iterative methodologies; their limitations when applying their practices; and their future directions are examined. Then the necessary tool support for agile methodologies, especially for Distributed eXtreme Programming (DXP) and distributed software development have been investigated. Available projects on distributed agile process tool support area have been investigated and these projects have been compared in order to clarify the necessities of distributed agile processes. Then with the findings of the previous comparison, the tool support need on software configuration management area has been examined and a study has been conducted on available open source software configuration management tools about their usability on agile processes. Finally, a software configuration management tool has been proposed in order to improve the efficiency and the usability of distributed agile processes, and the tool has been developed and evaluated in comparison with similar tools.

5.1 Discussion of Findings

With sufficient tool support people oriented incremental processes like agile methodologies can be executed on a virtual environment with distributed teams. In this study, the areas that necessitate tool support have been investigated and the existing ideas directed to the solutions have been presented. Collaboration of the team in case collocation cannot be realized, has been found out to be the most important problem, and the usage of tools that support collaboration in order to manage the team, conduct a team work during the development phase, and information exchange are stated to be the solutions for this problem.

The study of relevant literature shows that mostly coordination, planning and project tracking tools, which are built upon the web technologies that allow the coordination and communication of the team. In development area, there exist tools that increase the quality of the end product while speeding up the coding by helping the developers to find out the errors and to correct them. In the configuration management area usually version controlling systems are used, which can support teamwork, but they lack information management and collaborative work support.

Within the scope of this study, ASCM is proposed in order to enhance team work in the areas of collaboration by giving team members the ability to access each other using chat sessions or mailing without necessitating knowledge of other users' personal information (e.g., e-mail address). This tool supports teamwork and information exchange using code review sessions. With the teamwork, the quality of the end product improves and with the aid of information exchange code itself becomes self explanatory. Also this tool encourages users to work together as it supplies the awareness of other users' work.

5.2 Future Work

ASCM is evaluated with the findings of the investigations on studies conducted on Chapter 2 and compared with the similar open source tools. However; the usage of the tool on a real life distributed agile project will clarify the weaknesses of the tool in a professional setting and with the gathered information from the end user perspective necessary improvements can be conducted.

The integration of the tool with existing coordination tools like Xplanner or XPWeb may be investigated in order to create a complete tool support for distributed agile processes. Also, the integration of the tool with development support tools enabling continuous integration and test-driven development should also be investigated in order to improve the quality of the code besides the delivery of frequent information about the status of the code stability.

As the progress on information media and the technology continues, the usage of distributed development should be expected to increase. Also the successful projects that are developed by the open source community will increase the popularity of developing software using virtual communities. In that area there should be investigations on adoption of agile methodologies to open source projects using virtual software development environments.

The usage and the adoption of agile practices should also be investigated in order to be applied by the large organizations where restructuring of the organization might be risky or the cost of restructuring cannot be compensated.

REFERENCES

[1] Abrahamsson, P., Warsta, J., Siponen, M. T., and Ronkainen, J., 2003, “New Directions on Agile Methods, A Comparative Analysis”, 25th International Conference on Software Engineering, May 03 - 10, 2003, Portland, Oregon

[2] Aegis, http://aegis.sourceforge.net, Last Date Accessed: 10.04.2006

[3] Agile Alliance, http://www.agilealliance.com, Last Date Accessed: 10.04.2006

[4] Ant, http://ant.apache.org, Last Date Accessed: 10.04.2006

[5] AntHill, http://www.urbancode.com/projects/anthill/ , Last Date Accessed: 10.04.2006

[6] Arch, http://gnuarch.org , Last Date Accessed: 10.04.2006

[7] Blotner, J.A., 2003, “It's more than just toys and food: leading agile development in an enterprise-class start-up”, IEEE Agile Development Conference Proceedings 2003.

[8] Boehm, B., 2002, “Get Ready for Agile Methods, with Care”, IEEE Computer Magazine January 2002.

[9] Borland jBuilder, http://www.borland.com , Last Date Accessed: 10.04.2006

[10] Bowen, S., Maurer, F., 2002, “Process Support and Knowledge Management for Virtual Teams doing ASD”, Proceedings of the Workshop on Cooperative Supports for Distributed Software Engineering Processes held at the 26th IEEE Annual International Computer Software and Application Conference 2002.

[11] Bowen, S., Maurer, F., 2002, “Designing a Distributed Software Development Support System Using a Peer-to-Peer Architecture”, Proceedings of the Workshop on

Cooperative Supports for Distributed Software Engineering Processes held at the 26th IEEE Annual International Computer Software and Application Conference 2002, http://sern.ucalgary.ca/%7Emilos/papers/2002/BowenMaurer2002b.pdf , Last Date Accessed: 10.04.2006

[12] Chau, T., Maurer F., 2004, "Knowledge Sharing in Agile Software Teams", in Proceedings Symposium Logic vs. Approximation. Springer] http://ebe.cpsc.ucalgary.ca/ebe/attach?page=Root.Root.PublicationList%2FChauMaurer2004.pdf , Last Date Accessed: 10.04.2006

[13] Cockburn, A., Highsmith, J., 2001, "Agile Software Development: The people factor", IEEE Computer Magazine November 2001.

[14] Cohen, D., Lindvall, M., and Costa, P., 2003, "Agile Software Development, A DACS State-of-the-Art Report", http://fc-md.umd.edu , Last Date Accessed: 10.04.2006

[15] Cohn, M., Ford, D., 2003, "Introducing an Agile Process to an Organization", IEEE Computer Magazine June 2003.

[16] CruiseControl, http://cruisecontrol.sourceforge.net , Last Date Accessed: 10.04.2006

[17] CVS (Concurrent Versions System), http://www.cvshome.org , Last Date Accessed: 10.04.2006

[18] CUSeeMe, http://www.fvc.com , Last Date Accessed: 10.04.2006

[19] DARCS, http://abridgegame.org/darcs/ , Last Date Accessed: 10.04.2006

[20] Eclipse IDE, http://www.eclipse.org , Last Date Accessed: 10.04.2006

[21] Extreme Programming Web Site, http://www.extremeprogramming.org , Last Date Accessed: 10.04.2006

[22] Fowler, M., 2003, “The New Methodology”, http://www.martinfowler.com/articles/newMethodology , Last Date Accessed: 10.04.2006

[23] Highsmith, J., 2002 “What is Agile Software Development?”, *Crosstalk*, pp. 4-9, October 2002

[24] Highsmith, J., Cockburn, A., 2001, “Agile Software Development: The Business of Innovation”, IEEE Computer Magazine September 2001.

[25] Holz, H. and Maurer, F., 2002, “Knowledge Management Support for Distributed Agile Software Processes”, http://sern.cpsc.ucalgary.ca/%7Emilos/papers/2003/holz_maurer.pdf , Last Date Accessed: 10.04.2006

[26] HttpUnit, http://httpunit.sourceforge.net , Last Date Accessed: 10.04.2006

[27] IntelliJ IDEA, http://www.jetbrains.com , Last Date Accessed: 10.04.2006

[28] jUnit, http://www.junit.org , Last Date Accessed: 10.04.2006

[29] jWebUnit, http://jwebunit.sourceforge.net , Last Date Accessed: 10.04.2006

[30] Kelter, U., Monecke, M., Schild, M., 2003 “*Do we need 'agile' Software Development Tools?”,* p.412-430 in: Aksit, M.; Mezini, M.; Unland, R.: Objects, Components, Architectures, Services, and Applications for a Networked World (International Conference NetObjectDays, NODe 2002; Erfurt, Germany, October 2002; Revised Papers); Springer LNCS 2591

[31] Kircher, M., Jain,P., Corsaro, A., and Levine, D., 2001, “Distributed Extreme Programming”, XP2001 - eXtreme Programming and Flexible Processes in Software Engineering, Villasimius, Sardinia, Italy, May 21-23, 2001

[32] Lecture notes of course EE647 on Software Engineering by Semih Bilgen at Electrical and Electronics Department at Middle East Technical University.

[33] Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., Zelkowitz, M., 2002, "Empirical Findings inAgile Methods"

[34] Lycett, M., et al., 2003, "Migrating Agile Methods to Standardized Development Practice", IEEE Computer Magazine June 2003.

[35] Maurer, F. 2002,"Supporting Distributed Extreme Programming", Proceedings Agile Universe/ XP Universe 2002, Springer, 2002 http://sern.ucalgary.ca/%7Emilos/papers/2002/Maurer2002.pdf , Last Date Accessed: 10.04.2006

[36] Maurer, F., Martel, S., 2002, "Process support for distributed extreme programming teams", ICSE 2002 Workshop on Global Software Development.

[37] MILOS, http://wwwagr.informatik.uni-kl.de/~milos/ , Last Date Accessed: 10.04.2006

[38] MASE, http://ebe.cpsc.ucalgary.ca/ebe/Wiki.jsp?page=Root.MASE , Last Date Accessed: 10.04.2006

[39] Monotone, http://www.venge.net/monotone/ , Last Date Accessed: 10.04.2006

[40] MS Netmeeting, http://www.microsoft.com/windows/netmeeting/default.asp , Last Date Accessed: 10.04.2006

[41] Paulk, M. C., 2001, "Extreme Programming from a CMM Perspective", IEEE Software Magazine November/December 2001.

[42] Read, K., Maurer, F., 2003, "Issues in Scaling Agile Using an Architecture-Centric Approach: A Tool-Based Solution", Proceedings XP Agile Universe 2003, Springer 2003, p. 157-165.
http://sern.ucalgary.ca/~milos/papers/2003/ReadMaurer_b.pdf , Last Date Accessed: 10.04.2006

[43] Reifer, D. J., Maurer, F., and Erdogmus, H., 2003, “Scaling Agile Methods”, IEEE Software Magazine July/August 2003.

[44] Reifer, D: J., 2003, “XP and the CMM”, IEEE Software Magazine May/June 2003.

[45] Sangam, http://sangam.sourceforge.net , Last Date Accessed: 10.04.2006

[46] Schümmer, T. and Schümmer, J., 2001,“Support for Distributed Teams in eXtreme Programming”, http://www.darmstadt.gmd.de/concert/people/schuemmePrivate/xp00.pdf , Last Date Accessed: 10.04.2006

[47] Scrum Development Process, http://www.controlchaos.com , Last Date Accessed: 10.04.2006

[48] Stotts, D., Williams, L., Nagappan, N., Baheti, P., Jen, D., Jackson, A.,2002, “Virtual Teaming: Experiments and Experiences with Distributed Pair Programming”

[49] Subversion, http://subversion.tigris.org , Last Date Accessed: 10.04.2006

[50] Summaries of 1st, 2nd, and 3rd eWorkshops on Agile Methods at Fraunhofer Center for Experimental Software Engineering, Maryland, http://fc-md.umd.edu , Last Date Accessed: 10.04.2006

[51] Superversion, http://www.superversion.org , Last Date Accessed: 10.04.2006

[52] Wiki technology, en.wikipedia.org/wiki/Main_Page , Last Date Accessed: 10.04.2006

[53] Williams, L., Cockburn, A., 2003, “Agile Software Development: It’s about Feedback and Change”, IEEE Computer Magazine June 2003.

[54] xPlanner, http://www.xplanner.org , Last Date Accessed: 10.04.2006

[55] XPWeb, http://xpweb.sourceforge.net , Last Date Accessed: 10.04.2006

[56] EE647 Course, METU EEE Department, http://www.eee.metu.edu.tr/%7Ebilgen/ee647.htm , Last Date Accessed: 10.04.2006

[57] eSVN, http://esvn.umputun.com , Last Date Accessed: 10.04.2006

[58] JSVN, http://jsvn.alternatecomputing.com , Last Date Accessed: 10.04.2006

[59] SmartSVN, http://www.smartcvs.com/smartsvn , Last Date Accessed: 10.04.2006

[60] Subcommander, http://subcommander.tigris.org , Last Date Accessed: 10.04.2006

[61] TortoiseSVN, http://tortoisesvn.sourceforge.net , Last Date Accessed: 10.04.2006

[62] JavaSVN API, http://tmate.org/svn/ , Last Date Accessed: 10.04.2006

[63] Highsmith, J. A., 2000, "Adaptive Software Development: A Collaborative Approach to Managing Complex Systems", NY: Dorset House Publishing

[64] Ambler, S., 2002, "Agile Modeling: Effective Practices for Extreme Programming and the Unified Process", John Wiley & Sons Inc., New York

[65] Cockburn, A. 2002, "Agile Software Development", Addison - Wesley Professional

[66] Stapleton, J., 1997, "Dynamic Systems Development Method – The method in practice", Addison – Wesley

[67] Palmer, S. R., Felsing, J. M., 2002, "A Practical Guide to Feature-Driven Development"

[68] Poppendieck, M., 2002, “Lean Programming”, http://www.agilealliance.org/articles/LeanProgramming.htm, Last Date Accessed: 10.04.2006

APPENDIX A

SOFTWARE REQUIREMENTS SPECIFICATION REPORT

A.1 Introduction

A.1.1. Purpose of this report

This report specifies the requirements of Another Software Configuration Management Tool. This document will constitute the basis for the succeeding phases of the development of the software project.

A.1.2. Scope of this report

1.2.1. Section 1.3 presents an overview of the product.

1.2.2. Section 1.4 presents the Definitions, Abbreviations and Acronyms.

1.2.3. Section 2 includes detailed Use Cases of the software.

1.2.4. Section 3 presents System Specifications.

1.2.5. Section 4 presents Specific Requirements such as External Interfaces and Data Tables.

A.1.3. Scope of this Product

Software configuration management is a necessity in order to control and track production of software. Use of configuration management increases performance, communication and collaboration as it decreases the burden over developers and teams. It includes version controlling systems and supports collaborative work of developers. It warns users in case of file conflicts or changes at central repository.

In case of Agile Methodologies and Distributed Software Development, Software Configuration Management becomes an important factor as the development process is incremental, and geographical distribution causes management issues. At this point Software Configuration Management Tool becomes the common medium that developers use very often in order not to lose the track of the project. Also this medium should take care of security and controlling issues and could refuse the integrity breaking operations.

In general use there exist fully decentralized SCM tools, which enable the developer to clone the master repository, and commit and checkout changes with other clones using peer to peer networking. But in case of eXtreme Programming, incremental development uses approximately two weeks period, so instead of fully decentralized

SCM tools a central repository with web access would be better. Also, most of the configuration management tools do not support the communication between developers so information exchange cannot be realized. So collaborative tool support for configuration management tools encourages the developers to communicate and exchange information as well as work together when needed.

A.1.4. Definitions, Abbreviations, Acronyms

1. Software Configuration Management (SCM): “Configuration management is unique identification, controlled storage, change control, and status reporting of selected intermediate work products, product components, and products during the life of a system.” (Hass, A., Configuration Management Principles and Practice, Addison-Wesley, 2004).
2. Subversion(SVN): A code versioning tool similar to CVS(Concurrent Versions System), but has several enhancements over CVS, path and file removing, renaming, copying. http://subversion.tigris.org/
3. ASCM: Another Software Configuration Management Tool
4. Administrator: Tool and Resource Manager and/or Project Manager
5. Developer: User working on one of the Projects.
6. Repository: Source Code Base.
7. Checkout: The action of pulling codes from Repository.
8. Commit: The action of putting codes into Repository.
9. Conflict: Difference between two committed files at the same lines, which may cause problems.

A.2 Overall Description

A.2.1 General Interaction

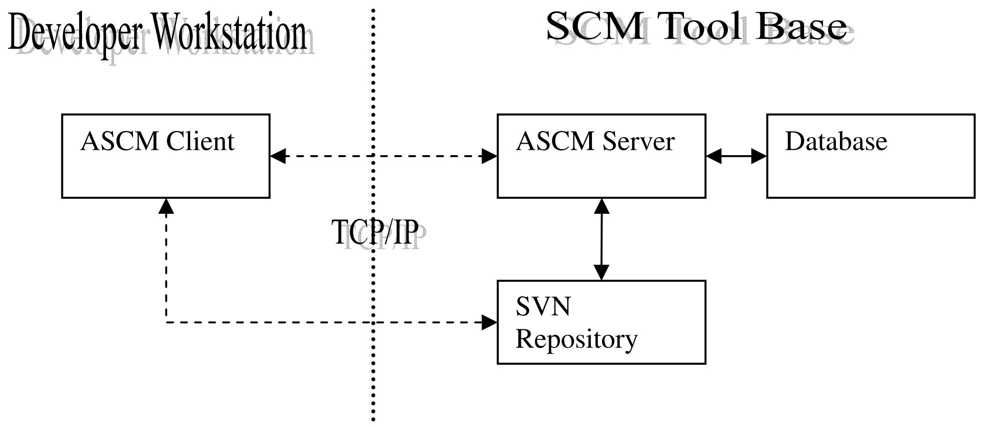


Figure 9 General Interaction Diagram

A.2.3 Product functions

Use cases

In this document use cases of SCM Tool are defined by tables.

Brief explanation of the table representation;

USE CASE #X→ The number of use case.	The name of use case
Goal in context	Defines the goal of function.
Preconditions	Defines from which condition does the function start.
Success end condition	Defines what will happen if the function succeeds.
Failed end condition	Defines what will happen if the function fails.
Primary and secondary actors	States the dealer/dealers of the function.
Trigger	Defines the starting point of the function.
DESCRIPTION→ Step by step description of the actions.	
1.	
2. etc.	
EXTENSIONS→ Definition of what else can be happen. **3** → Number of the step that this extension occurs. • → Definition of the extension. **Action:** → The action which will be done for the extension.	

USE CASE #1	LOGIN
Goal in context	A level of security and management issues enable the user to access user related data and do work at appropriate levels.
Preconditions	Main page is accessed via URL and

	displayed properly.
Success end condition	User starts a session with his/her privileges.
Failed end condition	A session cannot be started or related user data cannot be gathered properly.
Primary and secondary actors	Administrator or Developer
Trigger	Login link is accessed by the user.

DESCRIPTION

1. Login page is displayed.

2. User enters the necessary information shown as below.

- User type
    - Administrator
    - Developer
- Username
- Password

EXTENSIONS

2

- User approves the information

Action:

Open a session for the user.

Go to the page specified appropriate to the user type.

- User selects to Cancel task.

Action:

Return to the Main Page.

- Entered Username and/or Password is not valid.

Action:

Login page is displayed with a “Invalid Username or Password!” error on top of the page.

User is reprompted to enter the necessary information.

USE CASE #2	SIGNUP
Goal in context	A level of security and management

	issues enable the user to access user related data and do work at appropriate levels.
Preconditions	User has no account on system.
Success end condition	User creates an account for the system.
Failed end condition	Main page cannot be displayed.
Primary and secondary actors	Developer
Trigger	Sign up link is accessed by the user.
DESCRIPTION	
1. Sign up page is displayed with a User form.	
2. User fills up the form and submits. Necessary information is Username, email address, information about skills.	
EXTENSIONS **1** • User selects to Cancel task. **Action:** Return to the Main Page.	

USE CASE #3	APPROVE DEVELOPER
Goal in context	Administrator approves the accounts of newly registered developers.
Preconditions	Administrator should open a session using the Login link.
Success end condition	Approved developers are informed via mail.
Failed end condition	Failed to send mail or save changes or lost data.
Primary and secondary actors	Administrator
Trigger	Administrator selects to view list of registrations.
DESCRIPTION	

1. Administrator Configuration Page is displayed.
2. Administrator selects View Registrations link.
3. List of newly signed up users are viewed.
5. Administrator can either approve a registration or reject it thus removing the registration entry from the database.
6. After the registration is approved developer is added to the available developers list and notified via mail.

USE CASE #4	CREATE PROJECT
Goal in context	Administrator can create new projects. Manage the resources of the existing projects and configure project information.
Preconditions	Administrator should open a session using the Login link.
Success end condition	Changes made in configurations are saved in database.
Failed end condition	Failed to save changes or lost data.
Primary and secondary actors	Administrator
Trigger	Administrator selects to create/manage configuration.
DESCRIPTION	
1. Administrator Configuration Page is displayed.	
2. Administrator selects Create or selects an existing project from the listed projects.	
3. When Create is selected, project related information should be entered – Project name – Project information – Project repository base	
4. According to the entered information Tool – creates a database record for the project and saves its information. – creates repository by interacting the SVN.	
5. After the project is created or an existing project is selected Project configuration	

page is displayed with the selected Project's information and with the available developer list.

USE CASE #5	ASSIGN DEVELOPERS TO PROJECTS
Goal in context	Assigning developers to work in projects.
Preconditions	Administrator should open a session using the Login link.
Success end condition	Changes made in configurations are saved in database. Assigned developers are given the access rights to repository of the project.
Failed end condition	Failed to save changes or lost data.
Primary and secondary actors	Administrator.
Trigger	Administrator selects to Assign Developers to Projects link.
DESCRIPTION	
1. Administrator Configuration Page is displayed.	
2. Administrator selects Assign Developers to Projects link.	
3. Administrator selects a Project and a Developer in order to assign to the project and submits.	
4. Developer is notified via mail with the information of the ASCM server IP.	
5. Developer is given access rights to the projects repository, and developer is directed to the ASCM Client download page. Where the client aplication can be downloaded.	

USE CASE #6	CONNECT TO THE SERVER
Goal in context	Developer enters the IP of the server host, username and password into the related parts on Connection Pane in

	ASCM client.
Preconditions	User starts client application. User is already registered to ASCM server.
Success end condition	Client application connects to the server.
Failed end condition	Connection could not be established.
Primary and secondary actors	Developer
Trigger	Connect button on ASCM client's Connection Pane
DESCRIPTION	
1. ASCM Client is started.	
2. User opens the Connection Pane and enters: - IP address of the server host - Username - Password - Update period of the client.	
3. Tool connects to the server and checks the username and password from the information in the user database.	
EXTENSIONS **2** • User does not enter all necessary fields **Action:** A dialog window warns the user about the reason of the connection failure. **2** • Developer does not fill some mandatory field(s). **Action:** Prompt the appropriate error message for "Field should not be blank" on dialog window. **3** • Entered username and password does not match the registered users' information.. **Action:** Prompt the appropriate error message for "Connection failed" on dialog window.	

USE CASE #7	IMPORT INTO REPOSITORY
Goal in context	Developer can select and import files into the repository by clicking the Import button on the Operations Pane and selecting the files to import.
Preconditions	User logins to the tool using a valid Username and Password as Developer.
Success end condition	Selected files are imported into the repository.
Failed end condition	Import operation cannot be realized.
Primary and secondary actors	Developer
Trigger	Import Button from the Operation Pane.
DESCRIPTION	
1. Developer connects to the System using Client.	
2. Developer clicks the Import button to create a project base.	
3. Developer selects the files to be imported.	
4. Developer enters the related Commit Message.	
5. Tool imports the selected files to the repository as the project base.	

USE CASE #8	CHECKOUT FROM REPOSITORY
Goal in context	Developer can checkout from the selected project's repository either by entering revision, or tag info.
Preconditions	User logins to the tool using a valid Username and Password. User selects the local workspace location. If the selected folder is not under version control, client warns user to perform checkout. Or user clicks the Checkout button from Operation Pane.

Success end condition	Developer performs checkout from repository.
Failed end condition	Developer cannot perform checkout from repository.
Primary and secondary actors	Developer
Trigger	Checkout operation is performed, either automatically or manually.
DESCRIPTION	
1. Developer connects to the System.	
2. Developer selects the Working Space location.	
3. If the selected location is not under version control, client warns user to perform checkout.	
4. Developer approves the checkout operation or decides to perform checkout operation manually.	

USE CASE #9	COMMIT INTO REPOSITORY
Goal in context	Developer can commit files into the project’s repository.
Preconditions	User logins to the tool using a valid Username and Password. Local working space is compared with the repository and modified files are listed with a different color periodically. User can commit the modified files by selecting them from the file list.
Success end condition	Developer performs commit into repository.
Failed end condition	Developer cannot perform commit into repository.
Primary and secondary actors	Developer
Trigger	Commit operation from Repository Pane after selecting a modified file.

DESCRIPTION
1. Developer logins to the Tool.
2. Developer selects a modified file and wants to commit the file.
3. Developer enters the Commit Message.
4. Client performs commit operation. First client sends the server Commit Operation On Progress signal in order to prevent other clients to perform simultaneous commit operations. Then client sends the server to Commit Operation Performed Message, on which the server informs the other clients about the Commit operation and committd files.
EXTENSIONS **3** • Developer selects the Cancel the operation. **Action:** Operation is canceled. **4** • Tool can not interact with SVN. **Action:** Prompt the appropriate error message for "SVN connection problem".

USE CASE #10	INFORM USERS
Goal in context	Intelligent Messenger informs related Users in case of a change, conflict or an error occurs.
Preconditions	Project should be generated.
Success end condition	Related Users are informed about the condition via notifications through the client or if they are offline through e-mail.
Failed end condition	E-mail cannot be sent to the related Users.
Primary and secondary actors	-
Trigger	A condition of change, conflict or an

	error occurs.
DESCRIPTION	
1.Users are informed in case of following conditions: If users are working on the same file, they are informed about the situation and invited to code review and editing session. If a user commits a file, the other users are notified about the changed files and asked to perform update operation. If a conflict occurs, the last author of the file and the user who encounters with the conflict are informed about the situation via mail or client notifications.	
EXTENSIONS **1** • E-mail cannot be sent. **Action:** Schedule a job for sending E-mail at a later time.	

USE CASE #11	TAG OPERATION
Goal in context	User can select files to be tagged in order to create a branch or a release.
Preconditions	User should be connected to the system.
Success end condition	Tag operation on selected files performed successfully.
Failed end condition	Tag operation fails.
Primary and secondary actors	Developer
Trigger	User clicks the Tag button from Operations Pane.
DESCRIPTION	
1.Selected files are tagged with the user selected name.	
EXTENSIONS **1** • Tool can not interact with SVN. **Action:** Prompt the appropriate error message for “SVN connection problem”.	

USE CASE #12	COMPARE OPERATION
Goal in context	User can select file revisions to be compared in order to see the differences between two revisions.
Preconditions	User should be connected to the system.
Success end condition	The result of the diff operation is viewed in a window with changes indicated.
Failed end condition	Diff operation fails.
Primary and secondary actors	Developer
Trigger	User selects a file then performs compare operation with entering a revision from Repository Pane.
DESCRIPTION	
1.Selected revisions are compared and the results are shown in a windows.	
EXTENSIONS **1** • Tool can not interact with SVN. **Action:** Prompt the appropriate error message for "SVN connection problem".	

USE CASE #13	EXCHANGE MESSAGES
Goal in context	Users can message with other users of the project from Messaging Pane.
Preconditions	User should be connected to the system.
Success end condition	Users messages are sent to the other users or the indicated user.
Failed end condition	Sending message fails.
Primary and secondary actors	Developer
Trigger	User enters a message in the text field on Messaging Pane and clicks Send button.

	User invites another user to a chat session.
DESCRIPTION	
1.User can send messages to other users from common messaging area or user can invite another online user to a chat session.	
EXTENSIONS **1** • Message cannot be sent. **Action:** Prompt the appropriate error message for “Message failed to be sent.”.	

USE CASE #14	CODE REVIEW
Goal in context	User can invite another user to review a selected source file.
Preconditions	User should be connected to the system.
Success end condition	A window with text editor and messaging pane is opened for code review session, using which developers can comment, edit and change the source file as well a messaging with each other.
Failed end condition	Opening file or sending messages fail.
Primary and secondary actors	Developer
Trigger	User selects a user then invites him/her for reviewing a selected file from Messaging Pane.

DESCRIPTION

1. A window with text editor and messaging pane is opened for code review session, using which developers can comment, edit and change the source file as well a messaging with each other.
2. Each users' editing will be in a different color in order to prevent confusion.
3. At the end the user who started the session can save the changes and can commit the source file.

EXTENSIONS

1

- Tool can not open the source file.

Action:

Prompt the appropriate error message for "File cannot be opened".

1

- Tool can not send messages.

Action:

Prompt the appropriate error message for "Message failed to be sent."

3

- Changes cannot be saved.

Action:

Prompt the appropriate error message for “Changes cannot be saved.”

USE CASE #15	DISPLAY FILE HISTORY
Goal in context	User can see the history of a source file by using Show History Operation from the Repository Panel after selecting the source file.
Preconditions	User should be connected to the system.
Success end condition	History of the file is viewed in a Text Area below the file list.
Failed end condition	File history cannot be gathered.
Primary and secondary actors	Developer
Trigger	User selects a file then performs Show History Operation
DESCRIPTION	
1. User selects a file to review history, then performs the Show History Operation. 2. File history is viewed in a Text Area.	
EXTENSIONS **1** • Tool can not interact with SVN. **Action:** Prompt the appropriate error message for “SVN connection problem”.	

Use case flow diagrams

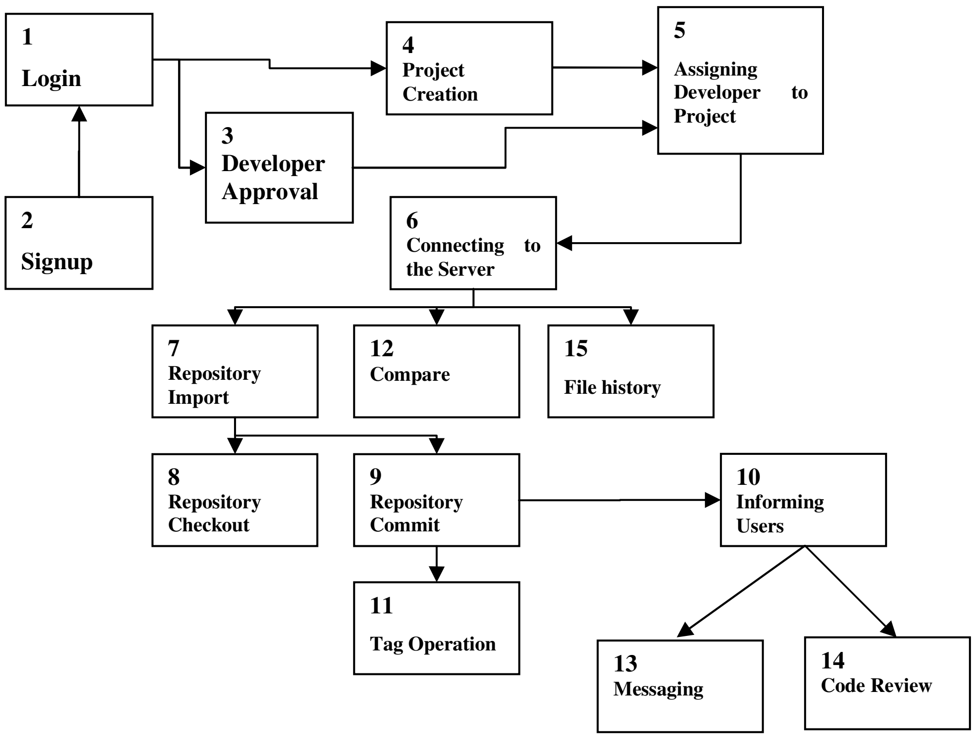


Figure 10 Use case flow diagrams

User characteristics

Name	Brief Description
Administrator	Development Leader Tool Administrator
Developer	Developer

Assumptions

- It is assumed that the users are able to connect the management portal via a Web browser using HTTP.
- It is assumed that port 1099 is open for connections.

A.3 System specification

Specification of the parts of the tool is presented below:

- Management Portal
  - Configurations of the tool users and management of access rights,
  - Repository management,
  - Project management,
  - Developer management.
- ASCM Server Application
  - Communication between client agents,
  - Used for checking local changes and informing related parties,
  - Collecting information over clients and generating necessary actions,
  - Used for handling Subversion base/client errors,
  - Uses notifications in order to inform related parties when necessary.
- ASCM Client Application
  - JavaSVN API Library is used for Subversion integration.
  - Used for the access of the Subversion base/client repositories.
  - Uses Notifications in order to inform related parties when necessary.
  - Used for performing standard Subversion operations
  - Generates automatic notifications when a change observed.
  - Enables developers to review and edit codes and then commit the changes after online chat sessions.
  - Enables messaging between developers.

A.4 Specific Requirements

External Interface requirements

Human interface Interactions

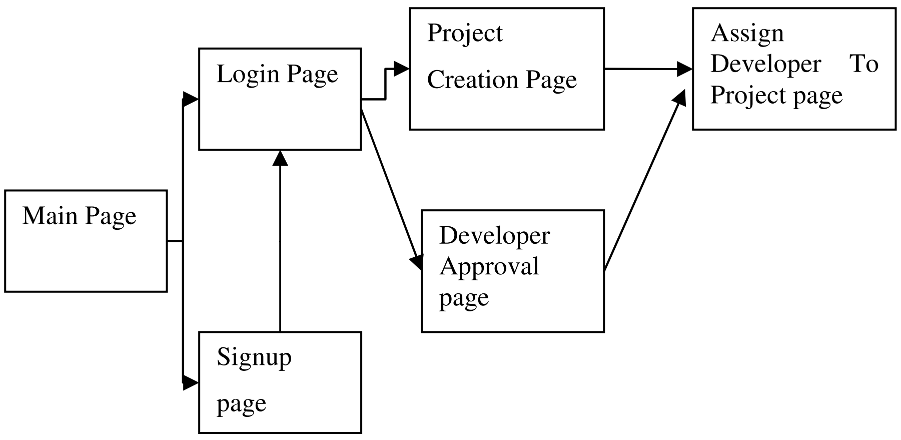


Figure 11 Portal interface interactions

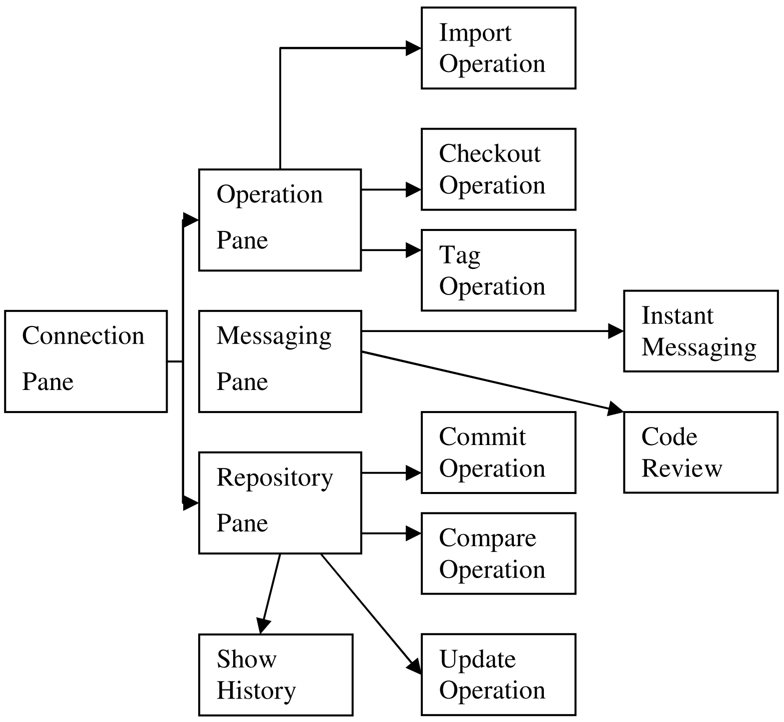


Figure 12 Client interface interactions

Data elements

In this document data objects of SCM Tool are defined by tables.

Brief explanation of the table representation;

Table Name	Name of the object defined by the table
Definition of table	Brief information about the table.
Data member	Name of the data member
Explanation	Brief definition of the data member
Foreign table no	Foreign table no that this data member is defined. (Reference to the table that the data object is defined)
Type	Type of the data C: char N: number D: date T: text P: Password
Length	Length of the data stored

- If the field in a table is an issue of design, it is left blank.
- If the field in a table is not relevant to the data member, it is filled with '-'.

Data element tables;

Table No	1			
Table Name	PROJECT_RECORD			
Definition of the data	record saved to the database belonging to a Project.			
Data member	**Explanation**	**Foreign table no**	**Type**	**Length**

PROJECT_ID	ID of the Project	-	N	-
PROJECT_NAME	Name of the Project	-	T	-
PROJECT_INFO	Information related to the Project	-	T	Not limited
PROJECT_BASE	Repository location of the Project	-	T	-

Table No	2			
Table Name	DEVELOPER_RECORD			
Definition of the data	record saved to the database belonging to a Team.			
Data member	**Explanation**	**Foreign table no**	**Type**	**Length**
DEVELOPER_ID	ID of the Developer	1	N	-
DEVELOPER_NAME	Name of the Developer	-	T	-
DEVELOPER_INFO	Information related to the Developer	-	T	Not limited
DEVELOPER_EMAIL	E-Mail Address of the User	-	T	-
DEVELOPER_PASS	Password of the User		P	6
PROJECT_ID	ID of the Project	-	N	-