

CubicProject

Integrere tester i et smidig prosjektstyringsverktøy

Jon Reinert Myhre

Master i datateknikk
Oppgaven levert: Juni 2007
Hovedveileder: Tor Stålhane, IDI

Oppgavetekst

I dag er det mange prosjekter som benytter smidige systemutviklingsmetoder som XP, Scrum eller Lean. Det har begynt å komme en del prosjektstyringsverktøy for å støtte denne typen prosjekter. Viktige momenter er korte iterasjoner, tettere samarbeid med kunde, god kravhåndtering og testdrevet utvikling, derfor bør et smidig prosjektstyringsverktøy ha god støtte for disse momentene.

CubicTest er et testverktøy som prøver å gjøre det enklere for ikke-tekniske personer å spesifisere tester for web applikasjoner. Det er ønskelig å koble CubicTest mot et open-source prosjektstyringsverktøy, bl.a for bedre å kunne koble tester til krav, integrere testene i prosjektets livssyklus, samt støtte flest mulig smidige praksiser i et prosjekt. Open-source er valgt siden CubicTest også er open-source.

Oppgaven gitt: 20. januar 2007
Hovedveileder: Tor Stålhane, IDI

ABSTRACT

With increasingly use of agile methodologies over the past years, there is also an increased need for testing strategies. Most of these testing strategies are the developers' responsibility. Acceptance testing however, is the responsibility of the customer, together with setting the requirements for the system being developed, or the project manager.

What the customer or the project manager often don't have information about, is the relationship between requirements to the system being developed, and the acceptance tests testing that the requirements are implemented correctly. This lack of overview makes it difficult to keep track of project progress. Since the information about project progress and the test results often is registered manually, this information may be incomplete or out of date.

This project has created a tool that provides traceability between tests and requirements, and automatic generation of requirement coverage reports on top of CubicTest. The tool is called CubicProject, and aims to increase the usefulness of CubicTest when creating acceptance tests for web applications, by providing a connection between tests and requirements. Before doing the implementation, feedback from the focus group held in connection with the project showed that the traceability was sought after by several different stakeholders.

Keywords: CubicTest, Selenium Exporter, JFeature, Agile Methodologies, Requirement Coverage, Acceptance Testing.


PREFACE

This master thesis documents the work done by Jon Reinert Myhre in the subject TDT 4900 Computer Technology, Master Thesis from January 20 to June 15, 2007. This master thesis is related to the CubicTest open-source project with focus on project management within agile development.

We would like to thank our supervisor Tor Stålhane at IDI, NTNU for guidance and invaluable feedback during this project, especially with the empirical part. We would also like to thank our co-supervisors Christian Schwarz at Bekk Consulting AS and Stein Kåre Skyttern at Comperio for their technical guidance, their indefatigable engagement and willingness to make CubicProject an interesting project to work with. In addition we would like to thank Erlend Halvorsen at IDI, NTNU for invaluable technical support and feedback and Hilde Skagestad at Computas for valuable feedback on both the contents and language in the report. We really appreciate the time and efforts they spent helping us during the project.

In the end we will also like to thank the participants in the workshop accomplished during the project. Without their time and effort, this project would have been difficult to complete.

Trondheim, June 15, 2007



Jon Reinert Myhre

CONTENTS

Introduction	1
1 Background and motivation	3
2 Problem definition	5
3 Project context	7
4 Research Agenda	9
4.1 Research Goal	9
4.2 Research Methods	10
4.2.1 Literature Study	10
4.2.2 Technical Study	10
4.2.3 Design and Implementation	10
4.2.4 Evaluation	10
5 Readers Guide / Project Outline	11
5.1 Introduction - Part I	11
5.2 Prestudy - Part II	11
5.3 Own Contribution - Part III	11
5.4 Evaluation - Part IV	11
5.5 Appendix - Part V	11
Prestudy	13
6 Agile Methodologies	15
6.1 Extreme Programming - XP	16
6.1.1 Values	16
6.1.2 Practices	18
6.2 Scrum	20
6.2.1 Pre-Game	21
6.2.2 Mid-Game	22

6.2.3	End-Game	22
6.3	Lean Software Development	22
6.3.1	Principles	23
6.4	The Crystal Methodologies	24
6.4.1	Properties	24
7	CubicTest	27
8	Project management and management tools	29
9	State Of The Art - Existing project management tools	31
9.1	Project Planning and Tracking System (PPTS)	31
9.2	XPWeb	31
9.3	XPlanner	32
9.4	BORG	32
9.5	Memoranda	33
9.6	activeCollab	33
9.7	Activity Manager	34
9.8	Summary	35
10	Distributed Storage of Data	37
10.1	Concurrent Version System (CVS)	37
10.2	Subversion (SVN)	38
10.3	Database Management System (DBMS)	38
10.4	Summary and Choice of Solution	39
11	Platform	41
11.1	Local-based (Standalone)	41
11.2	Web-based	41
11.3	Using the Eclipse Platform	42
11.4	Summary and Choice of Solution	43
12	The Eclipse Project	45
12.1	The Eclipse Rich Client Platform (RPC)	46
12.2	The Workbench	46
12.2.1	Standard Widget Toolkit	46
12.2.2	JFace	47
12.3	The Workspace	48
12.4	Team Support	48
12.5	Help	48
13	JFeature	49
	Own Contribution	51
14	CubicProject Design	53
14.1	Metaphors	53

14.2	User Stories	55
14.3	Domain Model	56
14.4	CubicProject Solution Using Eclipse	57
14.5	CubicProject Solution using ESB	58
14.6	Focus Group	61
14.6.1	Summary	62
14.6.2	Conclusion	64
15	CubicProject Solution	65
15.1	Implementation	66
15.1.1	selenium.runner	66
15.1.2	selenium.runner.holders	68
15.1.3	jfeature.core	68
15.1.4	jfeature.eclipse.common	69
15.2	CubicProject UI and User Documentation	69
	Evaluation	73
16	Discussion	75
16.1	The Prestudy	75
16.2	The Design Process	75
16.3	The CubicProject Tool	77
17	Conclusion	79
18	Further Work	81
	Appendix	83
A	Source code	85
A.1	selenium.runner	85
A.2	selenium.runner.holders	86
A.3	jfeature.core	88
A.4	jfeature.eclipse.common	90

LIST OF FIGURES

6.1	From values via principles to practices [1]	17
6.2	Scrum Process Overview [2]	21
9.1	The XPlanner layer architecture	33
9.2	The Activity Manager class diagram	35
12.1	The Eclipse Architecture	45
12.2	The Eclipse Workbench	47
13.1	The .jrj file where requirements are mapped to test methods	49
13.2	The Requirement Coverage Report	50
14.1	Metaphors	54
14.2	Domain Model	57
14.3	Eclipse Plugin Architecture	58
14.4	An ESB integrating a variety of disparate technologies [3]	59
14.5	Enterprise Service Bus Solution	60
15.1	Class Diagram For The Solution	67
15.2	Running Tests In CubicTest Using The Selenium Exporter	69
15.3	CubicTest Tests Finished	70
15.4	Generated Requirement Coverage Report	71
15.5	Detailed Requirement Coverage Report	71

LISTINGS

A.1	RunnerSetup Excerption	85
A.2	SeleniumHolder addResult Excerption	86
A.3	SeleniumHolder calculateResults Excerption	87
A.4	RequirementCoverageManagerImpl Excerption	88
A.5	JFeaturePlugin setProject Excerption	90

Part I

Introduction

This part serves as an introduction to the project. It begins with a chapter about the background and motivation for the project, followed by a chapter describing the problem definition. The two next chapters states the context of the project and the research goals and research methods that are be used. The last chapter serves as a readers guide for this thesis.

CHAPTER 1

BACKGROUND AND MOTIVATION

Many projects are today using system development methods like XP [4], [1], or Lean [5] and/or management methods like Scrum [6]. Aspects of importance are short iterations, closer co-operation with customer, good handling of requirements and test-driven development. Recently, some project management tools have arrived on the market, to support these kind of projects. It is therefore important that a project management tool for agile development has good support for one or several of the aspects mentioned above.

CubicTest is a tool to be used by customers and project managers (non-technical users) and developers (technical users) to create and run automatic acceptance tests for static and dynamic web applications, and also to model use cases or user-stories in CubicTest to generate basis code before starting to develop the application. It is important that the tool has high usability, so that the time the user spends on creating a correct test becomes as short as possible.

It is desirable to let CubicTest integrate with a project management tool to create an easy way to connect tests and requirements, integrate the tests in the project life-cycle and to support as many agile methodologies as possible. This integration will offer both customers/project managers and developers increased control over development of web applications, when using agile methodologies to develop them. It will become easier for the customer and project manager to follow the project progression at the same time as the developers and the project manager's understanding for the requirements will increase. The increased understanding will especially concern the order in which the requirements will be implemented, prioritizing within the various time-boxes and how the requirements specified in CubicTest interact on and affect each other.

A suitable project management tool should be based on open-source, just as CubicTest. With this integration, it will be easier for the customer representative and/or the project manager to understand the use and benefits of CubicTest and to follow the project progression.

CHAPTER 2

PROBLEM DEFINITION

The aim of this master thesis is to integrate CubicTest with an open-source project management tool based on agile methodology, to provide traceability between tests from CubicTest and requirements.

The project is divided into two parts. In the first part, we will evaluate several state-of-the-art open-source project management tools and specify certain requirements that should be present in a suitable tool. The aim for this part is to find a tool which satisfies the specified requirements and can be integrated with CubicTest. The integration depends on whether or not there exists a suitable tool among the state-of-the-art tools available.

The second part of the thesis depends on the first one. If it turn out that there exists a suitable tool, the aim will be to modify this tool, and modify CubicTest, and integrate them. On the other hand, if a suitable tool cannot be found among the available state-of-the-art project management tools, the aim is to specify a framework for a project management tool which can be integrated with CubicTest and, if there is enough time, implement a prototype of such a tool. The tool will be called “CubicProject”, based on it’s close interaction with CubicTest.

This tool, as a prototype of making use of test-results from CubicTest in a project management tool, should be a basis for developing the tool into a fully usable project management tool which shall be distributed as open-source software alongside CubicTest.

CHAPTER 3

PROJECT CONTEXT

This project is carried out as a master thesis at the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU) in Trondheim, with Tor Stålhane as main adviser. The project is an extension to the former master thesis [7] and [8], and is carried out in cooperation with Stein Kåre Skytteren at Comperio and Christian Schwarz at Bekk Consulting AS (BEKK).

Like CubicTest, this project is done in the context of agile development, and hence choices and decisions made during the project will be based on agile methodologies and the agile way of thinking. CubicTest has been empirically tested to be better than other state-of-the-art when it comes to usability, efficiency and quality [7]. Testing can be difficult without any evident relation to the requirements being tested. Integrating CubicTest into a tool where tests and requirements are related to each other will make the development project easier to manage.

CHAPTER 4

RESEARCH AGENDA

This chapter describes the goal of this thesis, and how it will be achieved. In section 4.2, research methods, the reader will find a description of this thesis' research methods. There will be a lot of work and planning between these parts, but it gives a picture of the workload and the areas that will be touched during the project.

4.1 Research Goal

The increasing use of agile methodologies within software development, has set more focus on the need for suitable project management tools to support agile development's adaptive and uncontrolled factors, and to help project managers or customer representatives to manage their software development projects in a better way. One of the main principles behind common agile methodologies is the need for rapid and frequent feedback. Feedback requires testing at all levels, including unit testing, regression testing, integration testing and acceptance testing. These first three test strategies are the developers responsibility and are done during the development. For these kinds of tests there exists a wide range of frameworks and tools.

Acceptance tests, on the other hand, are the responsibility of the customer since these tests tell the customer to what degree the finished product actually fulfills the customer's requirements. Hence, tools and frameworks for acceptance testing must be designed to let the customers and other non-technical persons easily take them into use. It was the lack of a user friendly state-of-the-art tool for acceptance tests to be used by non-technical users that triggered the development of AutAT¹.

To ease the project manager's and customer representatives' job, there is clearly a need for a project management tool that has the ability to integrate CubicTest or exchange information with it. Since CubicTest is the only one of its kind, there clearly doesn't exist a project management tool with the ability of GUI-based automatic acceptance testing of web applications. Hence, the research goal for this thesis is to either find a suitable project management tool that CubicTest can be integrated with, or develop a framework for a project management tool that CubicTest could be integrated with.

¹CubicTest's original name was AutAT - Automatic Acceptance Testing of Web Applications

4.2 Research Methods

4.2.1 Literature Study

A literature study will be done for several reasons. The literature study will cover agile methodologies in general and some of the most well-known agile methods in particular. This will be done to get a better overview of what agile thinking really implies, and how the values and principles are transformed into practices within the light-weight methodologies. The literature study must be done to achieve a better understanding of which properties that must be present in a project management tool in order to be useful for managing projects that use one or several agile methods to manage software development projects. The knowledge is also needed to achieve a better understanding of the importance of acceptance testing within agile development.

4.2.2 Technical Study

The technical study will in many ways cover the larger part of this thesis. First, several open-source project management tools will be tried out to see if there exists a suitable tool that can help us to achieve the aim of this thesis. We also have to dig deeper into CubicTest to see how it can be integrated with a project management tool. If a suitable tool cannot be found, several technologies must be studied to achieve a basis knowledge for how a framework for a suitable project management tool should be specified and designed to let CubicTest easily integrate with it.

4.2.3 Design and Implementation

This part will be conducted to do more research on how useful it would be to integrate CubicTest into a project management tool for agile software development, and if this integration actually increases project managers or customer representatives' ability to effectively manage agile software development projects. A solution shall be designed and discussed, and implemented if there is enough time.

4.2.4 Evaluation

The evaluation part will be done to evaluate the chosen tool or designed framework and integration with CubicTest. It will also contain some reflections and personal experiences regarding the research process. In addition, the conclusions drawn from the project and possible future work, will be presented here.

CHAPTER 5

READERS GUIDE / PROJECT OUTLINE

5.1 Introduction - Part I

This part serves as an introduction to the project. It presents the background and motivation, the problem definition and the context for this project. The two last chapters presents the research goals and methods used in the project, and this guide respectively.

5.2 Prestudy - Part II

The prestudy focus on agile methodologies, technologies and tools used in some way as input to the project. Topics presented are agile methodologies, project management, CubicTest, distributed storage and platform solutions, Eclipse and JFeature.

5.3 Own Contribution - Part III

This part holds the design solution of CubicProject, including proposed platform solutions and a discussion with domain experts, and the implementation of CubicProject.

5.4 Evaluation - Part IV

The focus in this part is on evaluation of the implemented system and evaluation of the project as a whole. It also contain the conclusions drawn from the evaluation and presents possible further work with the project.

5.5 Appendix - Part V

Here, the reader will find the modified code described in part III, that realizes the integration between CubicTest and JFeature.

Part II

Prestudy

This part holds the studying of all agile methodologies, technologies and tools used directly or indirectly as input to the project. The first chapter presents agile methodologies and some well-known agile methods, while the next two chapters presents project management and CubicTest respectively. Then a chapter where several state-of-the-art project management tools are studied followed by two chapters evaluating distributed storage technologies and application platforms. The last two chapters presents the Eclipse platform and the tool JFeature.

CHAPTER 6

AGILE METHODOLOGIES

This chapter explores agile methodologies in general and some of the most well-known agile methods in particular. This is done to provide more information about the context of this thesis, which is agile development. The study is also conducted to give better insight in important aspects and functionalities when studying different tools in chapter 9, and to be able to make more convenient decisions when implementing a solution.

Agile methods is a collective term of several light-weight development methods, which, in some cases, have replaced heavy-weight development processes as the Waterfall model¹. In many ways, agile methodologies are more like a philosophy than an approach, because it defines guidelines for developing software, rather than strict requirements².

Agile methods can be seen as adaptive methods rather than as predictive ones. This is due to that agile methods can adapt quickly to changing realities, for example modifications in requirements or modifications in the development team. On the other hand, agile methods will have difficulties to predict what will happen in the future. In addition, agile methods are more people-oriented than process-oriented. It is based on close collaboration, rapid feedback and human expertise, and not on complex document-centric processes [9].

The background for agile methods was that traditional approaches and methods/models turned out to be poorly suited for large-scale projects with an undefined or unknowable scope, and where the requirements frequently changed during development. It seemed to be a need for a faster and more effective way to develop software, that could handle these over-complex projects without large risks. Later, agile methods have also been adjusted to minimize risk, minimize documentation, increase communication among the developers and between the developers/project manager and the customer representative(s), emphasize early and continuous delivery, and appreciate changing requirements. The values behind agile methodologies are expressed in the Manifesto for Agile Software Development³:

The values below are used as basis for the principles behind the Agile Manifesto⁴. These principles need practices to be materialized as ways to develop software

¹http://en.wikipedia.org/wiki/Waterfall_model

²<http://sphereofinfluence.com/soiblogs/Tscheer/archive/2005/09/19/AgileVsLean.aspx>

³<http://agilemanifesto.org/>

⁴<http://www.agilemanifesto.org/principles.html>

and therefore, during the past years, several agile development processes has been brought to life, among others, eXtreme Programming (XP)⁵, Lean⁶ and Crystal Methodologies⁷. One of the most well-known agile management processes, Scrum⁸, also uses agile principles as a basis for project management.

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

Since CubicTest is developed in the context of eXtreme Programming, this lightweight development process will also be the main target of CubicProject.

6.1 Extreme Programming - XP

Extreme Programming is a lightweight methodology based on addressing constraints in software development. It can work with teams of any size and adapts to vague or rapidly changing requirements [1]. The aim of XP is to lower the cost of changes, ideally to keep it constant during the entire development process.

Extreme programming is based on a set of values, principles and practices. According to [4], XP has four values: communication, feedback, simplicity and courage. In [1], Kent Beck adds respect as a fifth value. These values are in many ways the invisible engine which makes XP work.

6.1.1 Values

Communication: It is vital to have good communication when practicing XP. XP assumes an active customer that participates daily during the project. While the developers are responsible for all technical decisions, the customer is responsible for business decisions. There is a mutual trust between the customer and the developers because the customer knows the developers have a better understanding for the technology, and the developers know that the customer is much more familiar with the problem domain. Both the customer and the developers trust each other that they are making the best decisions regarding their expert domain. In this way, they can answer questions from each other directly, which saves the project a lot of time, and increase the flexibility for all involved. This is of course how the communication will be in an ideal world, most real development environments are not that idyllic.

Feedback: By listening to and learning from all stakeholders, XP will provide rapid and frequent feedback, and changes can be done smoothly. This is important because the environment is changing continuously, and the team has to adjust accordingly. XP focuses on frequent planning, communication, design and testing to create continuing loops of feedback. Frequent planning feedback helps refining schedules,

⁵<http://www.extremeprogramming.org>

⁶<http://www.advanced-projects.com/Products/LPM.htm>

⁷<http://cjunpin.myweb.uga.edu/>

⁸<http://www.controlchaos.com/about/>

frequent communication feedback makes it easier to solve problems and discuss changes, and frequent testing feedback increases the confidence in making changes. Frequent release feedback also makes the customer believe in the project, and that the developers are on the right track. These feedback loops will make the team always ready for external changes.

Simplicity: Simplicity is a key word when performing XP - Sometimes called “the art of maximizing the amount of work not done” [10]. It pervades everything from only solving today’s problem today and only creating the simplest solution that could possibly work, to only building the system that needs to be built. Doing everything the simplest way together with feedback and communication, makes it easier to know what to do and to solve new problems.

Courage: In XP courage mean several things. If a problem arises when getting feedback, deal with it. On the other hand, only deal with the problem when you are certain of the cause. If you aren’t able to deliver before the release date, tell the customer without hesitating. If a solution can be simplified, do it. Courage alone can be dangerous and lead to over-confidence and actions without being aware of the consequences. Without courage, the other values will be hard to achieve but courage together with the other values will make the other values even stronger and help the project to become as successful as possible.

Respect: Respect is important when practicing communication, feedback and simplicity, and follows courage in every manner. One should have respect for other peoples knowledge, for the feedback from tests and for the fact that there often exist a simpler solution.

The principles are the “bridge” in XP, they translates the abstract basic values into practices. They are not directly human activities, but more like guidelines for how to take advantage of the values presented above. Kent Beck has defined a list of principles, the list can in its entirety be found in [1], chapter 5. What are much more interesting are the practices that the principles lead to. The practices will be thoroughly discussed below.

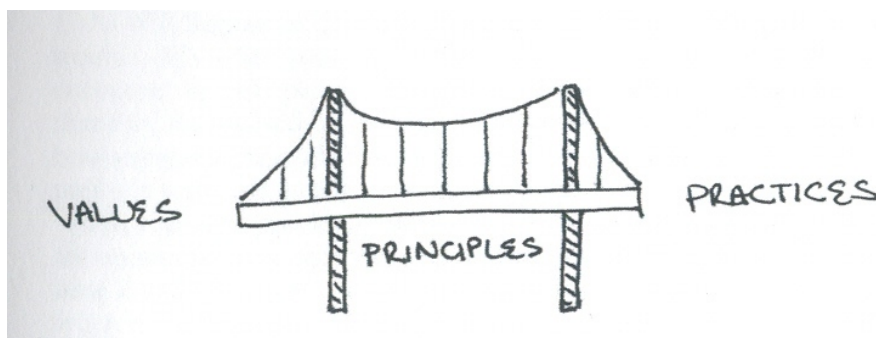


Figure 6.1: From values via principles to practices [1]

6.1.2 Practices

Sit Together: “Sit together” is a practice to fulfill the value of communication and feedback. If a team is sitting together, it will be much easier for the team members to communicate with each other, and with the customer representative. Sitting together gives the members the opportunity to communicate with all their senses, and there will be less room for misunderstandings when explaining a problem or a possible solution. The team will save a lot of time of walking, e-mail reading and e-mail writing, which they instead can use a small part of to communicate directly and the rest to get earlier finished with their work. This way the feedback will also be expressed directly, and the whole team will be better equipped for external changes. Principles in [1] that this practice relates to, could among others be humanity, improvement, flow and quality.

Whole Team: This practice is fundamental in XP. It is the whole team who develops the software. It is the whole team who owns the code together. It is the whole team who shares the responsibility. No team member stands above the others. This reflects the principle of diversity [1], which is in high degree an ideal principle. In the real world, there will always be someone with better skills than others, and they will often have more influence when making decisions.

Informative Workspace: The point here is to keep focus on the work, and to let any interested observer view information about the progress, problems and tasks without disturbing the team. In addition, the workspace should fulfill the team member’s needs while they are working, to reflect the principle of humanity.

Energized Work: This practice is also a central theme in XP. Work with a constant pace. Respect yourself and work when you are fit, not when you are tired or worn-out. Stay home and get well as quick as possible when you are ill instead of going to work because you feel you have to. You will not be able to work with the same pace when you are ill, and by showing up for work, you may risk the health of the other team members. When you are at work, do what you are supposed to do and don’t waste time on other things. Energized work is related to the humanity principle, which leads back to the value respect.

Pair Programming: It would be easy to think that using two people to produce one set of code would be a waste. But the fact is that having one programming and one other observing the code, looking for improvements and keeping track of the big picture, actually produces better code and faster. The difficult thing with pair programming is to learn to think aloud, to communicate thoughts. This practice is linked to the communication value through the principles reflection, redundancy and quality.

Stories: Stories are one way of XP to express simplicity. The intention of using stories instead of regular requirements is to focus only on needed functionality. This is related to the value simplicity - do the simplest thing that could possibly work. XP uses story cards to physically separate the different stories. This way a story card will contain one, and only on single needed functionality or feature or an improvement to one. The story cards are used for early estimation of the effort needed to implement the feature and to what cost.

Cycles: XP is an iterative methodology, and plans in cycles. Quarterly cycles are often used for high-level planning and at the end, reflection and evaluation. During quarterly planning, the focus is on the big picture. Here, stories for this quarter are chosen. Possible repairs and bottlenecks are also addressed. The project team should also make room for some slack by either adding some minor, low-priority stories, or reserve a week or to for other activities. The quarterly cycles can be split up into weekly cycles. This is related to the principle of baby steps and the aim of small but frequent releases. Activities in the weekly cycles are among others to review the progress, to prioritize and choose stories needed to be implemented that week, and further to separate the stories into smaller tasks.

Continuous Integration and Ten-Minute Build: Continuous integration is yet another way XP takes advantage of agility. Together with the ten-minute build they are practices that represent the baby-step principle. Integrate and test updates and changes as often as reasonable to integrate as few changes at once, not more than after a couple of hours. The more often you integrate, the more smoothly the integration and deploying of the software as a whole will run. The integration and testing should be automatic if possible.

Test-First Programming: or Test-Driven Development, as it is also called, is closely related to the principles of mutual benefit and reflection. It is also a good example of how the value of feedback is made visible in XP. When practicing test-first programming, everyone must ask themselves “what is the purpose of this feature or function”, and “why is it needed”, before they start to implement the feature. The next step is to write a test that explains why. Then, write an initial function the feature should contain, that fails when running the test. From there, improve the function until the test succeeds. This is how test-driven developments work like a safety-net in XP.

Incremental Design: Don't plan ahead. Do a little design each day, but always keep in mind to only do the simplest thing that could possibly work. Make changes when the need for them occur, don't save them for later. This way your work will always be up to date, the cost of changes will be as low as possible. The practice is a result from the principles improvement, quality, baby steps and economy, and is related to the value of simplicity. It is here worth mentioning that incremental design versus upfront design is a tradeoff the project manager and the developers has to make, based on what implies the least risk. Postponing decisions favors incremental design, but expected increase in implementation cost favors upfront work [11].

Extreme programming is managed by four variables; time, scope, resources and quality. Within more traditional methods of developing software, all four variables are likely to vary during the development process. With XP, the only variable that is allowed to vary is the scope. Before an iteration, the XP-team starts with an agreement on the quality level of the software being developed, and time and resources remains fixed during the iteration [4]. The measured time is ideal time, and velocity is the ratio between ideal and actual real time.

The iterations consist of one or several user stories or story cards, and the story are divided into smaller pieces - tasks. In XP, the work that will be done in an iteration, is decided with the Planning Game. The game can be carried out in two

ways, either the customer sets a release date for the iteration and add story cards until total number of ideal hours reaches maximum ideal hours for the iteration, or the customer chooses the story cards needed in the iteration, adds up the number of ideal hours and sets a release date based on this number [12].

This way, the team can always estimate how much that can be done during the iterations. If the stories are finished earlier than expected, new stories can be added up to maximum ideal time within the iteration. If the team doesn't manage to complete all the stories within maximum ideal time, because of e.g. changing requirements, stories can be removed from the iteration and added to e.g. the next iteration.

In addition, successful XP assumes a sufficient amount of external factors. This includes time, resources, constant cost of change, developer effectiveness and freedom to experiment [4].

6.2 Scrum

Scrum is an agile light-weight method for managing software development. The approach was first described by Hirotaka Takeuchi and Ikujiro Nonaka [13].

Scrum treats the development process as a controlled black box instead of a fully defined process. It assumes that the software development process is unpredictable and complicated, with several unknown aspects throughout the process and adopts an empirical approach by maximizing the ability to respond to changes, prior to understand and define the problem 100%. One of the key purposes of Scrum is to prepare for the unexpected.

The rise of Scrum was a need for a method that could manage certain problems detected when using older, more formal methodologies e.g. the Waterfall methodology [14]. The problems were requirements not fully understood in the beginning of a development process, changing requirements during the development process and the need for maximum flexibility and control during the development process [15].

Scrum makes developers able to operate adaptively within an environment that changes and evolve independent of the project. The management process for software development in Scrum is an enhancement of the iterative and incremental approaches, but it's not a linear process, compared to for example the misunderstood Waterfall model introduced by Winston Royce in 1970.

Scrum emphasizes a set of project management values and practices, rather than practices and values in requirements and implementation and so on. Easy combined with, or complimentary, to other methods. Scrum also uses six variables to plan software releases; requirements, time, competition, quality, vision and resource. These variables are part of the environment and do also change in response to increasing project complexity, and the increasing complexity leads to an increased need for control mechanisms. These control mechanisms, where risk management is one of the most important one, are used to increase the flexibility.

The common activities in Scrum, also known as the Scrum lifecycle is pre-game (planning and staging), mid-game (development) and end-game (release). As opposed to the more traditional linear methodologies, Scrum doesn't require these activities to follow each other in a fixed order when implemented, they can be used in whatever sequence most appropriate for the project⁹.

As mentioned above, Scrum is based on a set of values - commitment, focus, openness, respect and courage. As we see, these values are not very different from the values stated for XP. This is not a surprise, because, as the XP values, these are also derived from the agile values of software development, which favors individuals and communication, working software, customer collaboration and change responsiveness.

Scrum includes three main phases; planning and high-level design (pre-game), sprint-cycle (mid-game) and closure (end-game)¹⁰.

6.2.1 Pre-Game

The planning phase defines all inputs, outputs and processes for the project, and all high-level design decisions are made here. Possible activities can be prioritizing requirements, identifying available resources and planning a release date. This phase is accomplished rather quickly, because Scrum expects many of these parameters to change during the sprints [16].

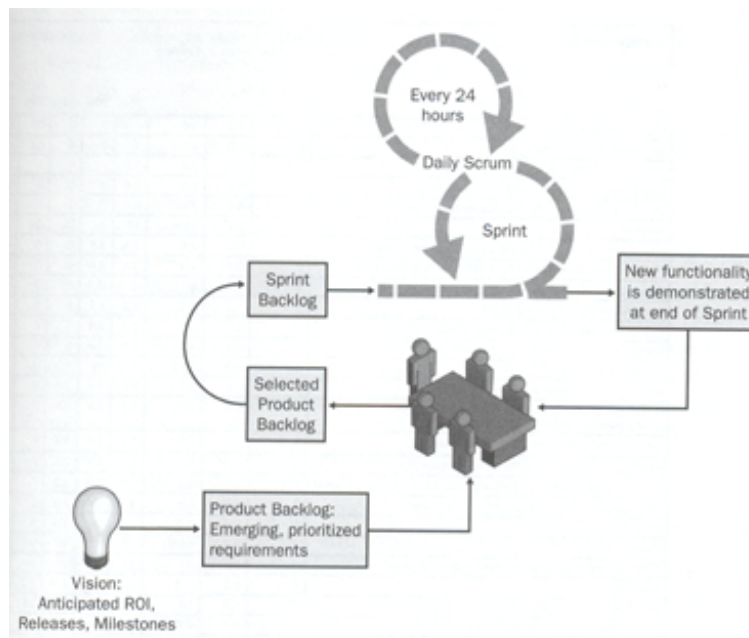


Figure 6.2: Scrum Process Overview [2]

⁹http://en.wikipedia.org/wiki/Scrum_%28development%29

¹⁰<http://www.codeproject.com/gen/design/scrum.asp>

6.2.2 Mid-Game

The Sprint-cycle phase is the heart of Scrum and is recognized as an iterative cycle of development work. It usually spans over a month, which is the period where the actual software development is done. A sprint always starts with a planning meeting to decide what to be done in the current sprint. Then the actual development is conducted before the sprint ends with a review meeting. The phase is treated as a black box with external controls to maximize flexibility while avoiding chaos. This way the final product evolves during the sprint. The sprint phase consists of the activities; develop, wrap, review and adjust, and can be conducted by one or several teams simultaneously.

Develop: Further development of the product in an iterative manner, including domain analysis, design, implementation, testing and documentation. Changes for the needed implementation could also be defined here.

Wrap: Closing the implemented “packet” or features, and make them ready for integration.

Review: Presenting work done in the sprint and review progress and risk. Issues and problems must also be raised and resolved here.

Adjust: This is a final step where possible changes to the work done in the planning phase could be addressed, regarding design, requirements or resources.

6.2.3 End-Game

The closure phase marks an end for the development. Until this point, the project is open to changes in the environment variables. Common activities in this phase are debugging, promotion and marketing. The phase ends with a deliverable, which closes the project, and the product is released. When this events occurs, depends on the controls used during the sprint phase.

6.3 Lean Software Development

Lean software development is Lean manufacturing applied directly to the software development domain. Lean manufacturing is a management methodology focusing on reduction of, or eliminating, all wastes, and to improve material handling, inventory, quality, scheduling, personnel and customer satisfaction^{11, 12}. In other words, a management approach for streamlining any production system¹³.

The main principles of Lean manufacturing, like waste reduction, continuous improvement and flexibility, are identical to some of the agile principles, which make Lean software development a natural member of the agile methodology family. Further we will present the seven principles of Lean software development, mainly based on Mary and John Poppendiecks’ explanations in [5].

¹¹http://en.wikipedia.org/wiki/Lean_manufacturing

¹²<http://www.strategosinc.com/principles.htm>

¹³<http://sphereofinfluence.com/soiblogs/Tscheer/archive/2005/09/19/AgileVsLean.aspx>

6.3.1 Principles

Eliminate waste: Anything that does not add value to a project, perceived by customer. Plan ahead, find out what the customer wants and deliver exactly that. Whatever gets in the way of rapidly satisfying the customer can be defined as waste. Waste can be partially accomplished work, extra processes or features or waiting caused by lack of communication.

Amplify Learning: Software development is a continuous learning process. The best way to improve software development is to amplify learning. The solution is among others, good feedback loops in all parts of the development process, especially provided by frequent oral communication and testing feedback at all levels, reasonable iteration planning, and good synchronization of the different parts of the project.

Decide As Late As Possible: Delaying decisions until as late as possible, makes it easier to base the decision on facts rather than on speculations. When delaying decisions until they are absolutely needed, the future is closer and easier to predict. This can be achieved, among other things, by concurrent development and must be obtained to as large degree as possible, together with a thorough judgment of all options and develop a sense when decisions must be made.

Deliver As Fast As Possible: Make the development cycles as short as possible. Faster and shorter cycles will make you learn more, because you will have more frequent reviews and improvements. Speed also assures that the customer gets what he needs when he needs it and not when his requirements have changed. This can be obtained by no delay in approving requests, staffing, clarification of requirements, testing, integration or deployment. Other keywords are as frequent cycles as possible without causing lack of learning, bad decision making and good development schedules.

Empower The Team: No one understands the details better than the people doing the work needed to be done. Let these people, guided by a leader; make the process and technical decisions. Due to delayed decisions and fast execution, the “pull” scheduling mechanism, from an upstream supplier onto a downstream consumer, is Lean’s way of schedule work and the local signaling mechanisms is how Lean practices to let the workers communicate with each other to know what needs to be done.

Build Integrity In: Conceptual and perceived integrity. Conceptual integrity is how the central parts of the system flow together and is an important part of the perceived integrity which is how the customer perceives the system’s usefulness over time. To maintain the conceptual integrity, good refactoring and testing routines are important aspects.

See The Whole: How to focus on overall system performance, rather than let each part of the development team suboptimize their own area of expertise, while still keeping the integrity in the system. This is even a bigger problem when companies are joining together to develop software. Make sensible contracts for all the elements and activities associated with the project, but make them as agile as possible.

6.4 The Crystal Methodologies

The Crystal Methodologies is a family of methodologies, suitable for projects with varying circumstances [9], and the principle creator is Alistair Cockburn. It is basically the size of the project and the degree of how critical the system is, that decides the appropriate methodology within the family [17]. The members of the family are indexed by colors to indicate the project “heaviness” that the different methodologies are suited for. The “heaviness” is decided by the team size, where larger teams require more resources and heavier methodology. Crystal assumes co-located teams.

The color index is graded so that bright colors means smaller teams and smaller project, and dark colors means larger teams assigned to larger projects. The index goes from Clear (8 or fewer people) via Yellow (10-20 people), Orange (20-50 people), Red (50-100 people), Maroon¹⁴ (100-200 people) and Blue (200-500) to Violet (500-1000 people) [18], [17]. Whereas Crystal Clear is most similar to the other methodologies described earlier, it will be the focus here.

Crystal Clear is most useful for small teams and designed for non-critical projects. As the other light-weight methodologies, Crystal Clear does also focus on people, not processes and artifacts.

The core guide to successful Crystal Clear development is to follow seven safety properties. Following these properties, Crystal Clear will be a way to prioritize *Safety* in the project outcome - the methodology aims to increase the probability of a successful delivery, *Efficiency* in development and *Habitability* of the conventions [18]. According to [19] the first three are mandatory. The properties described below are based on [18] and [19].

6.4.1 Properties

Frequent Delivery: Frequent delivery is used to provide feedback. The sponsors will get feedback concerning the progress of the team, the users get a chance to provide feedback to the team concerning implemented requirements, and the developers gets a moral boost to keep up the good work until the next delivery.

Reflective Improvement: In short, reflective improvement is obtained when the team get together and agree on what’s working and what isn’t, discuss possible improvements or changes and at last realize those improvements or changes in the next iteration.

Osmotic Communication: The main point here is to make the team members able to pick up oral information in an osmotic kind of way. This could be realized by letting the whole team work in the same room, and let questions and answers to problems flow in the background hearing of the team members and let everyone pick up relevant information and decide for themselves if they want to contribute to the discussion or not.

¹⁴Several articles are using Magenta instead of Maroon, but since Cockburn uses Maroon in [18], we will focus on that.

Personal Safety: Personal safety has nothing to do with physical safety, it is an early step toward trust. The point is to speak without fear for others' reactions. This can be achieved by always giving constructive criticism to other team members and always speak in a friendly manner. If the personal safety is in place, honesty can be achieved and honesty can provide trust. Trust makes everybody feel more committed to the project. Lack of personal safety could lead people in the team to retain information or even lie. In any case, this could provide seriously damage to the communication within the team.

Focus: Focus means that all team members focus on the work they are supposed to do, the work that provides business value. This requires guaranteeing the team members some time without letting them be distracted or disturbed by meetings, customer request, incoming e-mails and telephones, and non-topic discussions. According to [18], two hour work with 100% focus two days in a row each week should be sufficient for any project.

Easy Access to Expert Users: This is also a matter of feedback. If expert users are available to the team, preferably on-site, the team will get rapid feedback on possible requirements, quality and design decisions, and the finished product much quicker, than if expert users only are available for instance once a month. This of course, supposes that the team are able to make this change decisions all by them selves. Weekly or semiweekly meetings with the expert users or to let developers gradually become expert users can be a solution.

Technical Environment with Automated Tests, Frequent Integration and Configuration Management: Automated tests increases feedback and quality, and saves time for further development. Configuration management provides asynchronous check in, check out, wrap up and roll back of code, and let the team provide developed code both separately and together. Frequent integration is also a feedback issue, the more often the team integrates the system, the more often they will be able to detect errors and mistakes. In addition, it is easier to rectify a small part of integrated code than a large part.

Crystal Clear offers a selection of strategies and techniques to satisfy the properties described above. On the other hand, Crystal Clear doesn't require any of the strategies or techniques in particular, if other strategies or techniques seem more appropriate, they should be used.

CHAPTER 7

CUBICTEST

An important part of the agile methodologies described in chapter 6 is testing. As mentioned, there are several types of testing, among others unit testing, integration testing, acceptance testing and regression testing. The test method that is most difficult to automate, is acceptance testing, especially of web applications. Acceptance testing of web applications is to a large degree done manually by customers or developers, or partly automated where developers have implemented test-code.

The following three paragraphs are taken from [20]. CubicTest¹ is a tool to be used by customers (non-technical users) and developers (technical users) to create and perform automatic acceptance tests for static and dynamic web applications, and also to model use cases or user-stories in CubicTest to generate basis code before starting to develop the application. It is important that the tool has high usability, so that the time the user spends on creating a correct test becomes as little as possible.

The background for bringing CubicTest to life was the increasing use of Test Driven Development (TDD)² and automatically testing at all levels, which has been introduced by XP and other agile methodologies. The purpose of these agile development methods is to let the customers create and maintain acceptance tests, and also to use the tests to specify new requirements. The core in TDD is to use tests to specify functionality, in other words to create tests before the application code. However, tools suitable for customers were a commodity in short supply, and resulted in acceptance tests created by the developers.

As a consequence of this, it became difficult for customers to specify new requirements based on the tests. This proved the need for an application tool that could relatively easily let non-technical persons, as well as technical persons, create tests.

Originally CubicTest was developed during a master thesis in the spring of 2005 as a tool for automatic acceptance testing of static HTML pages, under the name AutAT [7]. It was tested to verify that it was better than available state-of-the-art open source tools for acceptance testing. During another master thesis in the spring of 2006, the tool became further developed to also create tests for dynamic web pages based on Java script or Ajax [8]. During this project, CubicTest was also extended to export tests through Watir³. Watir, or Web Application Testing in Ruby, as it stands for, is an open-source automated testing tool which can be used when

¹<http://www.cubictest.org>

²http://en.wikipedia.org/wiki/Test_driven_development

³<http://wtr.rubyforge.org/>

performing functional testing, automated acceptance testing or large-scale system testing on web applications using Internet Explorer.

Up to this day, the tool has got a few more upgrades and in addition to this project, Erlend Halvorsen at IDI, NTNU is also writing a master thesis about the tool [21]. A major change is that CubicTest no longer uses the Watir-exporter developed in [8], but a Selenium-exporter, based on Selenium. Selenium⁴ is another framework for testing web application functionality. It is platform independent, browser independent, and provides two types of tests; Browser compatibility testing and system functional testing. The Selenium-exporter walks through every step in the test, and run calls into Selenium. Both this, and the Watir-exporter are built as exporter architecture, which mean that anyone can use them to develop their own exporters to export their own tests. CubicTest are available at OpenQA.org⁵, where it is licensed under the GNU General Public Licence⁶.

⁴<http://www.openqa.org/selenium/>

⁵<http://svn.openqa.org/fisheye>

⁶<http://www.gnu.org/licenses/>

CHAPTER 8

PROJECT MANAGEMENT AND MANAGEMENT TOOLS

A project is an one-time event to create a unique service or product within defined scope, time, and cost constraints. The essence of project management is organizing and managing resources to let the resources complete the project within the defined constraints. To do this, several activities has to be carried out. The activities are for example planning work, assigning tasks, progress reporting, and quality management.

Common for all of them, and for all the others which hasn't been mentioned, is that they require some form for documentation. All this documentation must be stored somewhere, not only in the head of the project manager. To minimize duplication and to make it easier to keep overview of all the plans, schedules and logs, it has, especially during the digital revolution, been more common to use tools to manage projects.

Far from everyone are convinced that using agile methods is the way to develop software. Much of the criticism is aimed against the fact that agile methods aren't predictive, that they are unstructured and undisciplined, that they produce too little documentation to be controllable and that they are difficult to adopt. To meet this criticism and to supplement agile methods as a better way to develop software compared to more traditional methods as the Waterfall or the iterative model, several project management tools has been developed during the past few years.

The purpose of CubicTest is to let the the customer set rapid feedback, by letting him/her see if the stories and tasks that was defined prior to the implementation, are implemented the way they were intended, and that they aren't changed or forgotten. CubicTest tests every single specified piece of a web application, without linking them up to the tasks and stories specified by the customer. To let a project management tool collect information from CubicTest about the fulfillment of the tasks, the tool must know which of all the pieces tested in CubicTest belongs to the tasks or requirements specified in the project management tool.

CHAPTER 9

STATE OF THE ART - EXISTING PROJECT MANAGEMENT TOOLS

This chapter contains a study of several existing project management tools, primary for developing web applications. The study is conducted with the purpose of finding a suitable tool that make us able to fulfill the goal for this project. If such a tool cannot be found, the study will give valuable insight in aspects and features required for a tool to be appropriate for integration with CubicTest in the context of agile methodologies, presented in chapter 6, and to be able to provide us with the wanted functionality, specified in chapter 1.

Important parameters that are valued, are loyalty to agile methodologies, information storage, information export and import possibilities, user friendliness and whether the focus is on management (hours, todo-lists, appointments and so on) or development (requirements, stories, work progress, test coverage and som on). Since CubicTest is an open-source application, the study will only contain open-source management tools. However, if elements from non open-source tools are used as basis of comparison, they will be referenced.

9.1 Project Planning and Tracking System (PPTS)

PPTS is a web-based open-source project management system, which supports the XP developing methodology. It is written in PHP and JavaScript, and uses MySQL as its main database. The tool can hold users and projects and assignments between these. The project can be split up into iterations, with options for Work Breakdown Structure, Progress Report, Metrics, Burn Down Graph and so on. In the Work Breakdown Structure, the user stories and the underlying tasks are defined and assigned. The project manager can also generate backlog of stories.

In addition, PPTS can hold information about room reservations, competences overview, absence and resource allocation. Particularly the possibility of registering absence in terms of sickness, holidays and courses/seminars is rather well developed.

9.2 XPWeb

XPWeb is also a web-based open-source project management system, which is written in PHP and JavaScript. It uses MySQL as distributed database and supports projects using the XP methodology to develop software, but focuses more on XP than

PPTS. XPWeb does not hold a significant amount of user information comparable to PPTS, but has a more well arranged overview of the users and their project assignments. In addition, the manager is able to add and/or edit user profiles in XPWeb, which isn't possible in PPTS. As in PPTS, one can also here add iterations, with belonging user-stories and tasks. It is also possible to register single user-stories and tasks.

Instead of assigning people only to projects as in PPTS, XPWeb also assigns people to tasks. This way, it is clear who is responsible for each task. One feature which is supported in XPWeb, but not in PPTS, is XP's use of pace and load factor regarding ideal days versus real days. XPWeb also have a calendar that shows the tasks and iterations, with filter options and the possibility of uploading documents related to each project. Because XPWeb consider work pace and load factor, it doesn't provide the possibilities of absence registering, resource allocation and competences overview like PPTS. XPWeb support export of data to XML¹.

9.3 XPlanner

XPlanner is an open-source project management tool written in JavaScript (js) and Java Servlet Pages (jsp). It is thus web-based and aims at managing projects using the XP methodology. XPlanner split projects into iterations and further into stories and tasks, but has a more complicated system when it comes to set estimated hours, than XPWeb. In addition, it doesn't take into consideration work pace and load factor like XPWeb does.

Like PPTS, XPlanner also have predefined roles, which are non-editable. XPlanner provides the possibility of assigning users at all levels. It supports distributed information token, time tracking and time sheet generation, generation of team velocity and individual hours metrics, charts for iteration velocity, adding attachments to the stories and tasks and have a large support for project information export to, among others, XML, MPX, PDF and iCal. It also has the ability to import user stories of the .xml format.

A layered architecture is used to build XPlanner. On top, JSP, CSS and HTML are used to present the user interface and format the planning domain objects, which are provided and manipulated by the "business logic" layer. These domain objects are not communicating directly with the database, but are loaded as Java objects using Hibernate². Hibernate accesses the data, which are stored in a MySQL database, using JDBC³.

9.4 BORG

BORG is an open-source tool which is a combination of a calendar and reminder system, and a task tracking system. It is written in Java, and can be used together

¹<http://www.w3.org/XML/>

²<http://www.hibernate.org/>

³<http://java.sun.com/javase/technologies/database/>

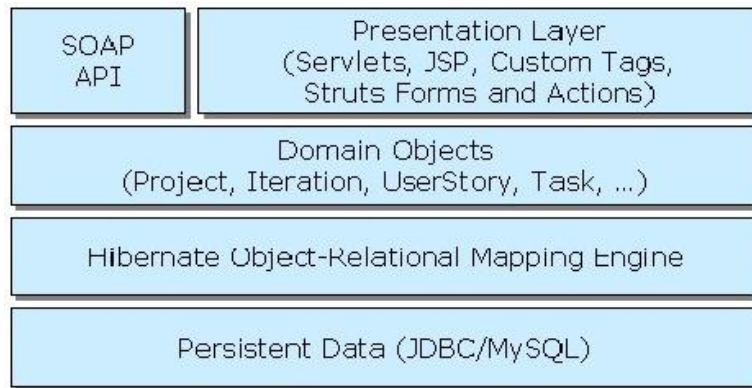


Figure 9.1: The XPlanner layer architecture

with for instance MySQL or HSQLDB. BORG works on both Windows and Linux. The tool is not based on a particular development methodology, but uses tasks as the primary way of registering information. The calendar part supports many sorts of appointments, including a to-do list. The task tracking part includes information regarding all tasks. A task can include subtasks and log their status, when they are opened and closed, and other changes in the main task. The tool also provides import and export of XML with task information.

9.5 Memoranda

Memoranda (formerly known as jNotes2) is an open-source diary manager and project scheduling tool. It is also a cross-platform application and written entirely in Java, and is thus available on any Java-enabled platform. The tool is designed for personal use, with personal settings and preferences in a multi-user environment. It is not project management groupware and doesn't support responsibility sharing and workgroup organizing. It is not designed with the intention of being used as a management tool for software development, although it can be used in this way, and thus doesn't support any development methodologies, such as XP or Scrum.

Managing projects is Memoranda's main feature. A project can hold information about documents, events, resources, tasks and agenda. Project diaries can be exported to HTML for web-site display. All information is transparently saved when switching features within a project. The task defines the actual project work and the user is able to update the progress on his/her tasks. A to-do list holds the progress overview of all tasks.

9.6 activeCollab

activeCollab is an open-source web-based tool for project management. It is mainly written in PHP and JavaScript, and uses a MySQL database for storing project information. The tool is well designed for team management with user login and several access and privilege levels. activeCollab's main feature is the set

up of projects, although it doesn't support any methodologies regarding software development.

A project can include messages, task lists, milestones and files. The tasklists are where the tasks are defined and team members are assigned to the tasks. Tasks can either be assigned to one single person or to a single company (group). There is no progress overview of the tasks, a task is either pending or closed. On the other hand, the tool holds an overview over all activities in the project with time and date.

9.7 Activity Manager

Activity Manager is a local-based tool for project management. It is open-source and designed to help project managers to keep an overview of and administrate project activities. Written in Java, Activity Manager can run in three different modes. Standalone mode with embedded HSQL, Server mode with MySQL or a user defined database using JDBC⁴ (custom mode).

The tool has built in support for export and import of database files in the XML format. It also has the ability to export data to EXCEL. In addition, Activity Manager includes a report engine. This is based on the Jakarta Apache Velocity⁵ and generates output in HTML based on template files. The tool register tasks and information related to these tasks according to a set of rules⁶:

- A task that has one or more subtasks can not accept any contribution
- A task that has one or more subtasks represents an aggregation of its subtasks
- A task that is associated to one or more contributions can not admit any subtask
- A task that is associated to one or more contributions has updatable 'Initial fund', 'Initial consumption' and 'Todo' fields
- Sister tasks must have different codes

The storing of information is based on an entity-relationship model, with four basic notions. These are collaborators (people who contribute to tasks), tasks (defined activities), durations (of contributions) and contributions. Contributions could be a single contributor, a single task, a duration or a date. The model is shown below, for further details, see⁷.

⁴<http://java.sun.com/javase/technologies/database/>

⁵<http://velocity.apache.org/>

⁶<http://www.jfbrazeau.fr/main/homepage/activitymgr/userGuide.do#CHAPTER2>

⁷<http://www.jfbrazeau.fr/main/homepage/activitymgr/userGuide.do>

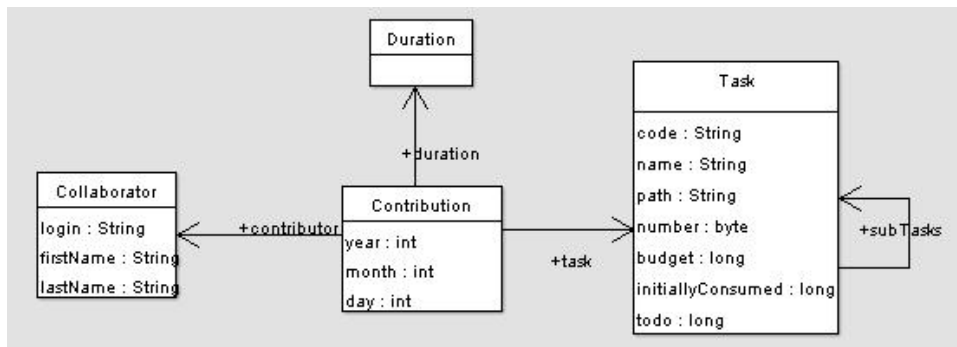


Figure 9.2: The Activity Manager class diagram

9.8 Summary

In this chapter, we have described seven tools for project management. The tools have some similarities. They are all open-source and made with the intention of being used by project managers to gain control over their project. They all store the project information and they have the opportunity to connect users to projects or project artifacts.

The simplest tool to use is Memoranda. Memoranda is a personal scheduling tool, and thus doesn't offer any distributed saving of data. In addition to manually managing projects with task progression, Memoranda provides a calendar feature with a daily agenda for managing other events in addition to projects. Not supporting any software development methodology and not using any distributed database makes it poorly suited for the purpose described in chapter 2. In addition, Memoranda is difficult to install for a non-technical user.

BORG Calendar has the same Calendar focus as Memoranda. What makes BORG better is the use of a distributed database and the ability to export XML-files, and that it's easier to install. BORG also has a more detailed way of handling information with the ability to add subtasks to the tasks. As for Memoranda, BORG lacks support for software development methodologies, and this also holds for activeCollab and Activity Manager. Both activeCollab and Activity Manager offers the ability to store project data in a distributed database and should both be possible to set up for non-technical users.

The three other tools we have looked at, PPTS, XPWeb and XPlanner, are similar. They are all designed with the purpose of being used for managing projects using Extreme Programming as the methodology for developing software, but doesn't support other agile methodologies. They are web-based and with the option of creating users with different roles, but XPWeb is the only tool where additional roles can be added, and existing roles can be modified. In PPTS and XPlanner, the roles are predefined and non-editable.

XPWeb is slightly more faithful to the XP-methodology than XPlanner and PPTS, particularly because it provides the possibility to set load factor and work pace in addition to separate ideal hours from actual hours. All the three tools use MySQL as

a distributed database for storing project data. Both Xplanner and XPWeb support export of data to XML, but XPlanner also supports import of user-stories of the .xml format.

This walk-through shows us that none of the tools are entirely what we are looking for. Some of them aren't designed with the intention of being used as a project management tool for software development at all. Some of the tools that support light-weight methodologies, are designed specially for being used with Extreme programming. All the tools lacked one or several desirable features e.g. not using distributed database or not supporting several agile methodologies. On the other hand, the study has helped us to point out features and functions a good tool should offer, to fulfill our intention.

- The tool should be user-friendly. It should be easy to install and set up, and only provide necessary features and functions.
- The tool should provide traceability from test to requirement.
- The tool should support agile methodologies in general, without favoring any method.
- Project data should ideally be stored distributed, not locally, see chapter 10.
- Import and export of information to and from an external tool seems inconvenient, thus the tool should be integrated into Eclipse and build on the Eclipse framework as a plug-in, due to its close relationship with CubicTest, and future integration with other services.
- The tool should provide a view over project progression and test status, and also all previously changes done to tests or the project.
- The tool should provide an opportunity to see who's responsible for the project, and also to assign people to tasks and requirements. This way, the project manager will always know who's doing what.

CHAPTER 10

DISTRIBUTED STORAGE OF DATA

How information should be stored, is an important aspect to consider when designing a tool for project management. The information regarding tests, test results and requirements should be easy to access for several stakeholders. In addition, it is interesting, especially for the project manager and the customer, to see how this information change over time. “Did this test run successfully last week?” or “how has the requirement coverage percent changed per week since the beginning of the project?” could be possible problems to be addressed. Distributed storing of data can be done in several ways. In this chapter we will briefly explain some of the most common state-of-the-art solutions.

10.1 Concurrent Version System (CVS)

CVS is a version control system, used to record and keep track of the history of source files and documents. CVS is open-source and an important component of Source Configuration Management (SCM)¹. It is licensed under the GNU General Public Licence².

CVS provides a view over all changes made to the files including time and date, what changes that is done, who has done the changes, and the possibility of rolling back to previous versions in case something is wrong with the latest change.

CVS uses a client-server architecture with a server that holds the current, latest updated, version or versions of the file(s) and connected clients that checks out the current version(s), make changes to the file(s) and later checking in these changes to update the version(s) on the server. To allow several people to work on the same files simultaneously and avoid conflicts, CVS only allow changes made to the latest version on the server. Still conflicts occur, usually because a user has forgotten to check out the latest version before he or she make changes to it.

To solve this problem, CVS offers the possibility for the user to manually merge his or her changed file(s) with the current version(s) on the server. Successful check-ins increment the version numbers on all involved files, which together with the authors name, date and a description line written by the author, are written to a log file by the server.

¹<http://www.eecs.wsu.edu/cs224/notes/scm.html>

²<http://www.gnu.org/licenses/gpl.html>

CVS have some limitations. First of all, renaming files and moving them aren't versioned. This also holds for symbolic links in files or documents. In addition, CVS provides limited support for Unicode text files and non-ASCII filenames, because CVS primarily run on Unix systems, and they run in UTF-8³.

10.2 Subversion (SVN)

SVN is an free/open-source version control system, released under the Subversion License. It is designed with client-server architecture from the beginning, as a modern replacement for CVS and hence offers many of the same features as CVS, in addition to features that CVS doesn't support. SVN can be used to manage source code along with any other files, but is not a software configuration management (SCM) system [22].

Regarding historical view of changes made to files and the possibility of rolling back to previous versions, SVN is just as good as CVS. In addition, SVN offers Apache network server option with WebDAV/DeltaV⁴ protocol, which makes SVN able to use the HTTP-based WebDAV/DeltaV protocol for network communications and provide repository-side network service with the Apache web server. This enables users to easily browse the repository containing the file versions. For those who don't want to run the Apache web server, SVN offers a standalone server option that uses a custom protocol. The repository can either be created with BerkleyDB as an embedded database back-end or back-end of normal flat files using a custom format.

SVN offers true versioning, which means that moving and renaming of files and folders are also versioned, not only altering the contents of files. SVN also provides versioning of metadata and symbolic links. This gives SVN an advantage over CVS.

10.3 Database Management System (DBMS)

An alternative to version control systems, is a traditional DBMS. A DBMS is designed for managing databases and consist of software that controls the organization, retrieval and storage of data in the database. Well known DBMSs are among others Oracle, MS Access, MS SQL Server, PostgreSQL and MySQL. All these are based on the relation model⁵, which is a database model based on first-order-logic⁶ and set-theory⁷. In addition, a DBMS include data structures optimized to deal with large amounts of data stored permanently and possibly distributed, a specified query language for altering the database and transaction rules that ideally guarantees the ACID properties⁸.

DBMSs are primarily designed for storing information, but files can also be stored. Versioning must be implemented manually if needed. MySQL is one of the most

³<http://www.utf-8.com/>

⁴<http://svnbook.red-bean.com/en/1.0/apc.html>

⁵http://en.wikipedia.org/wiki/Relational_model

⁶<http://www.answers.com/topic/first-order-predicate-calculus>

⁷<http://plato.stanford.edu/entries/set-theory/>

⁸http://www.service-architecture.com/database/articles/acid_properties.html

popular open source DBMSs. MySQL is a multithreaded, multi-user database management system, using SQL as query language. It is owned and sponsored by MySQL AB⁹, which distributes it freely, develops and maintains the system and selling support to it.

MySQL are a popular DBMS used together with web-applications, especially web-applications written in PHP (Hypertext Preprocessor)¹⁰.

10.4 Summary and Choice of Solution

In this chapter, three ways of storing distributed data are described. These are probably the most common state-of-the-art methods of storing data in a distributed manner. CVS and SVN are similar in their functionality, SVN providing some additional features. Common for them both is that they are versioning control systems, they store files and keep track of all changes done to the files and by who, and they provide functionality for rolling back to previous versions of the files.

Database Management Systems (DBMSs) are a different way of storing data. Although databases can be used to store files, the primary goal of databases is to store raw information in an easy accessible way, optimized for dealing with large amounts of data, and with transaction rules to guarantee the ACID properties.

Our goal with CubicProject is primarily to provide traceability between tests and requirements when developing software. The way CubicTest works today, it stores test files locally. When tests are run, it generates a popup message which isn't stored at all. What we primarily want to store are not the tests themselves, but only information about the result from the tests, which requirements the tests covers, and to what degree it covers the requirements. In addition it could be useful to store information about who alters the tests and when.

This information should be easy accessible for the project managers. It doesn't seem useful to use versioning systems for this, since it would require us to collect all this information and pack it into files, and then store the files. This solution doesn't make the information as easily accessible as we want. It would be more appropriate to store the information in a database, since in this way our data would be stored in an easy accessible way, and hence be easy to present to the project managers or other stakeholders. In addition, to store and update the information in a database, requires less knowledge than using versioning systems like CVS or SVN.

⁹<http://www.mysql.com>

¹⁰<http://www.php.net>

CHAPTER 11

PLATFORM

There are several platforms to choose among, each one of them with their own advantages and disadvantages. In this chapter we will look at some possible platform solutions for CubicProject.

11.1 Local-based (Standalone)

A local-based client can be built in many ways. Aspects to be considered here are installation effort and possible features. Some advantages for a stand-alone solution:

- Keyboard shortcuts (hotkeys) can be implemented.
- Support for cut & paste can easily be provided.
- Support for drag & drop functionality can be implemented.
- Large degree of freedom regarding possible features and design of the application.

Some disadvantages:

- Software must be distributed and installed on all user's computers.
- Software changes and updates must be distributed.
- Installation manual should be provided.

11.2 Web-based

Aspects to have in mind when considering a web-based client are availability and distributed storage of data. Arguments for a web-based client:

- No installation or installation guide are needed.
- The software can be used by a large number of users simultaneously.
- All users store results from requirements and test coverage in the same place and tests can easily be viewed or modified by other users.
- Version control of test coverage can be managed easily.

- The software can be used independent of operating system.

Some disadvantages:

- Poor support for cut & paste.
- Difficult to implement support for drag & drop functionality.
- Dependent on network and network speed when used and when interacting with CubicTest.
- Limited options regarding possible features and GUI design.

11.3 Using the Eclipse Platform

The Eclipse platform is in many ways more similar to a local-based platform than a web-based one. On the other hand, Eclipse is an IDE¹ and hence offers a lot of features and functions. Some advantages of building CubicProject as an Eclipse plug-in:

- Easy to interact with CubicTest, since CubicTest runs on the Eclipse platform.
- Gets features and functions from Eclipse and from other Eclipse plugins for free.
- Support for keyboard shortcuts (hotkeys).
- Can easily be integrated with other Eclipse plugins.
- Possible to implement support for drag & drop functionality.
- Many developers are familiar with Eclipse.

Some disadvantages:

- Can be difficult for non-technical users to set up and configure.
- Software and software updates/changes must be distributed and installed on all user's computers.
- Options regarding possible features and design of the application are restricted to Eclipse.

¹Integrated Development Environment

11.4 Summary and Choice of Solution

The three platform solutions described in this chapter all represent different categories. As extremities, we have the standalone solution and the web-based solution, which in many ways stand as opposed to each other. Both the standalone and the web-based solution are solutions that have to be built from scratch, with the pros and cons this involves.

Using the standalone solution makes us able to customize the solution as much as necessary, but also requires most effort to develop. To develop a web-based solution will also require a lot of work, and compared to a standalone solution, the web lays some restrictions and limits the options regarding possible features and functions to implement. However, a web-based solution can be accessed remotely from anywhere, a local-based solution must be installed wherever it is intended to be used.

Somewhere between the standalone and the web-based solution, we find the Eclipse solution. Using the Eclipse platform will give us a standalone solution, but provide us with a large number of features and functions already implemented in, or available as plugins to, the Eclipse framework.

As stated in chapter 4 and with respect to the agile methodologies described in chapter 6, our primary goal is to provide traceability between tests in CubicTest and requirements. Since CubicTest is built upon the Eclipse platform, it seems like a poor solution not to use the advantages by creating CubicProject as an Eclipse plugin. Using the Eclipse platform to create CubicProject, will make us able to integrate CubicTest the easiest way, and thus make the traceability clear and easy to follow. We will be dependent on neither network bandwidth, nor a standalone solution that has to run in addition to Eclipse. Using Eclipse will provide us with the same wide range of GUI features and functions that are used when developing CubicTest.

CHAPTER 12

THE ECLIPSE PROJECT

Since it was concluded in section 11.4 that Eclipse is best suited as a platform for the CubicProject solution, it is useful and necessary with a little more insight in how the Eclipse platform works and how it is built up. Being more familiar with the Eclipse technology will also be an advantage when the solution shall be designed and implemented. This chapter is an introduction to the Eclipse Project which produces the Eclipse SDK. A complete technical description can be found in [23].

The Eclipse Project is a open-source software framework primarily written in Java. It is platform independent and runs today on most operating systems, including Microsoft Windows, Linux, different Unix flavors and Mac OS X. It consists of the four sub-projects Equinox, Platform, Java development tools (JDT), and Plug-in Development Environment (PDE). What is important here, is the Platform project, which holds the key features in Eclipse and provides the core framework and services upon which all plug-in extensions are created.

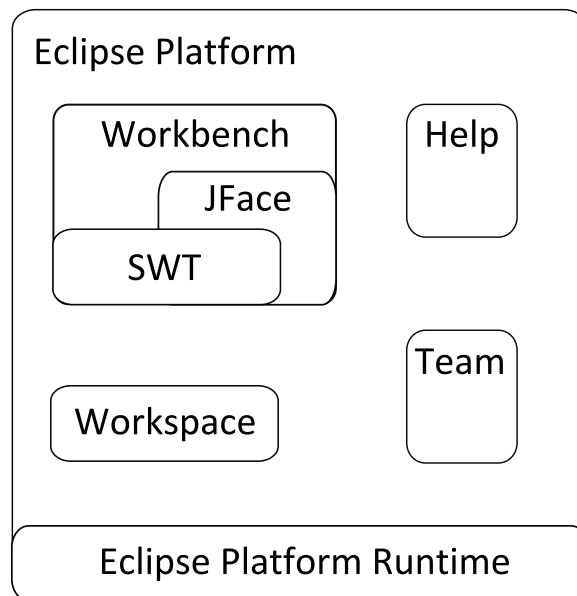


Figure 12.1: The Eclipse Architecture

The Eclipse Platform, shown in figure 12.1, builds on the Eclipse Platform Runtime where all plugins are loaded, integrated and executed. The Platform Runtime is

the core of the Eclipse Platform, booting up Eclipse - creating an instance of the Runtime, which again is responsible for starting all other parts of Eclipse, e.g. plugins that will provide the desired functionality.

The Eclipse Platform is structured as a foundation onto which it is possible to develop and integrate a rich set of tools as plugins. Plugins are structured bundles of code and/or data that contribute functionality to the system. Plugins can define *extension points*, which are well-defined places where other plugins can add functionality, and in this way each sub-system in the platform is structured as a plugin or set of plugins that add functionality. Functions can also be added as code libraries or documentation.

Other parts of the Eclipse Platform is the *Workspace*, which is the area of files and directories that a user want to work on with Eclipse, and the *Workbench* which provides the user interface of Eclipse and belonging key features.

12.1 The Eclipse Rich Client Platform (RPC)

The Eclipse RCP is known as the minimum set of plugins to develop a rich client application with a User Interface (UI). In addition to constituting the Platform Runtime (*org.eclipse.core.runtime*), the RCP also holds the *org.eclipse.ui* plugin, known as the Workbench.

12.2 The Workbench

The Workbench plugin provides the UI structure for Eclipse and defines a number of extension points for other plugins contributing to the UI, e.g. menu and toolbar actions, drag and drop operations, dialogs and wizards, and custom views and editors. The Workbench is responsible for the presentation and coordination of the user interface. An example of the UI of a running instance of Eclipse, is shown in figure 12.2. What the Workbench shows is the contents of the active *Workspace*.

The Workspace consist of one or more projects which is shown in the navigator view in the upper left part of the figure. Contents of files in a project can be edited and manipulated in the *Editor*, as shown in the upper right part of the figure. The Editor is the main part of the workbench, and can have specialized behavior for certain file types.

In addition, the workbench consists of two other views, which can show the outline of a file, tasks or problems, errors or javadoc. If a plugin provides functionality to Eclipse and has its own editor, the editor and corresponding preferred views can be put together and loaded as a *perspective*. The Workbench plugin also includes the JFace UI framework and the Standard Widget Toolkit (SWT).

12.2.1 Standard Widget Toolkit

The Standard Widget Toolkit (SWT) is a low-level, platform independent toolkit that supports platform integration and provides a common API with standard widgets

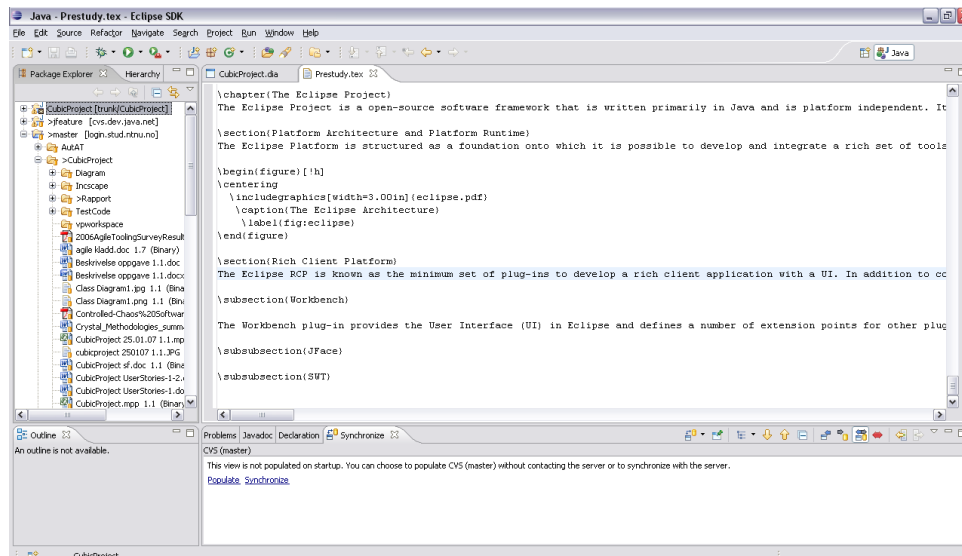


Figure 12.2: The Eclipse Workbench

that a developer can use to make applications look as native as possible to the current running operating system. The SWT is responsible for all information presented to the user through the Eclipse Platform User Interface (UI). The SWT combines features offered by the traditional Java Abstract Window Toolkit (AWT) which provide low-level widgets such as list, text fields and buttons and the Java Swing toolkit which emulates widgets like trees, tables, and rich text not supported by Java AWT, in addition to provide a look and feel emulation layer that attempt to make applications look similar to the underlying native window system.

The problem is that there exists a gap between the emulated widgets and the native widgets. SWT tries to address this problem by defining a common API where SWT prioritize to use native widgets wherever possible for each native window system, and provides a suitable emulation if there is no native widget available. This API is available across a number of supported window systems.

12.2.2 JFace

The JFace UI framework works with SWT and provides higher-level application constructs, in other words classes that provide a simple way of handling many common UI programming tasks. In addition to include image and font registries, dialog, preference, wizard frameworks, and progress reporting in the UI toolkit components, JFace also provides *actions* and *viewers*.

The action feature provides independently defined user commands regarding their exact whereabouts in the UI. Actions know their own key UI properties and represent a command that the user can trigger with a button or a menu item. The viewers are model-based adapters for certain SWT widgets. Viewers has the ability to provide higher-level semantics than what is available through the SWT widgets and can handle common behavior.

12.3 The Workspace

As mentioned in section 12.2, the active workspace is shown by the workbench. The workspace is a designated area on the file system which contains a collection of files and directories the user are working on, organized as projects. The projects can have specified natures regarding their purpose. This way the project can call on applicable plugins to give the project certain properties. E.g. the standard Java project nature includes a classpath within the properties, and a plugin project includes instructions for how to read the manifest file.

12.4 Team Support

The Team Support allows version and configuration management for files and folders in a project. Plugins allow other plugins to define and register implementations for shared programming, configuration management, repository access and versioning. An extension point from the Team API is the only information a repository provider needs when he/she builds a new plug-in to provide access to a new type of repository. There exists plugins for the most common team repositories, including CVS and SVN.

12.5 Help

This plugin implements a platform optimized for help web server and document integration facility, and defines extension points which can be used by other plugins to contribute plugin help or documentation. Raw documentation is contributed as HTML files and are organized as online books with separate XML files containing suitable navigation structures. Small tools usually store XML navigation files and HTML content files together with the code, while larger tools separate them into a code plugin and a help plugin.

CHAPTER 13

JFEATURE

JFeature is an open-source feature/requirement coverage tool that facilitates focusing on requirements as you develop code¹. Originally, the tool was named JRequire and worked on top of JUnit, and JFeature is only some minor adjustments added to JRequire. JFeature uses JUnit² to test code and identify the requirement coverage based on results from the unit tests. It is built as an Eclipse plugin, and hence GUI and other features are based on existing Eclipse technology. JFeature is licensed under the Eclipse Public License³.

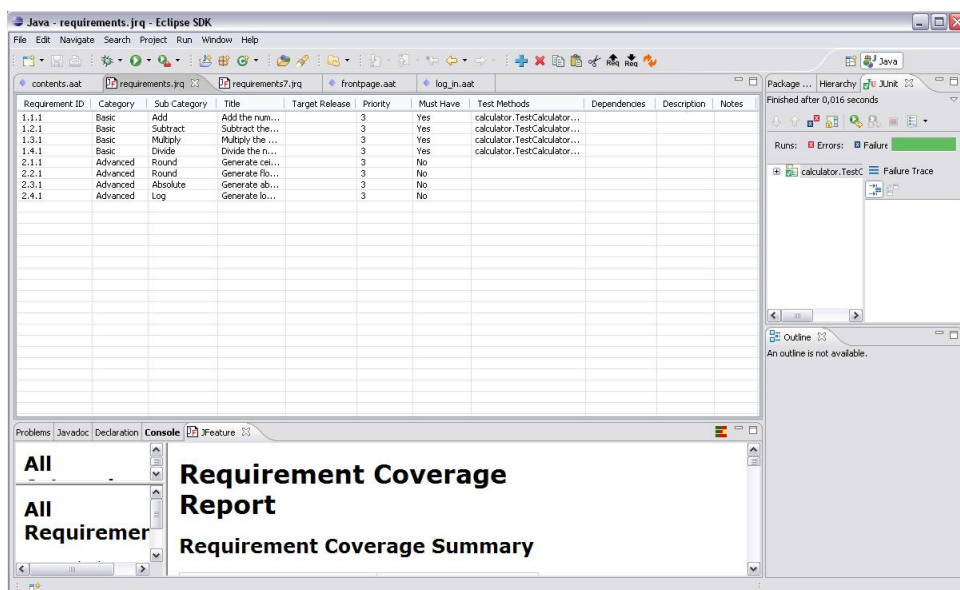


Figure 13.1: The .jrj file where requirements are mapped to test methods

As an Eclipse plugin, the intention with JFeature is to be easy to use during development, and let the user with a little effort keep the requirement coverage updated during development. It has support for keeping the requirements updated when refactoring the unit tests. In addition, JFeature supports both import and export of requirements either as .cvs files or as .xml files. JFeature presents the requirements

¹<http://www.technobuff.net/webapp/product/showProduct.do?name=jfeature>

²<http://www.junit.org>

³<http://www.opensource.org/licenses/eclipse-1.0.php>

and the mapping to unit tests in Eclipse as .jrj files. In the .jrj file the user can either import requirements from .csv or .xml files and choose which test cases to map each requirement with. The requirement file(s) has to be associated with a project containing JUnit tests, hence the requirements can only be mapped to test cases within the selected project. On the other hand, several .jrj files can be associated with the same project simultaneously.

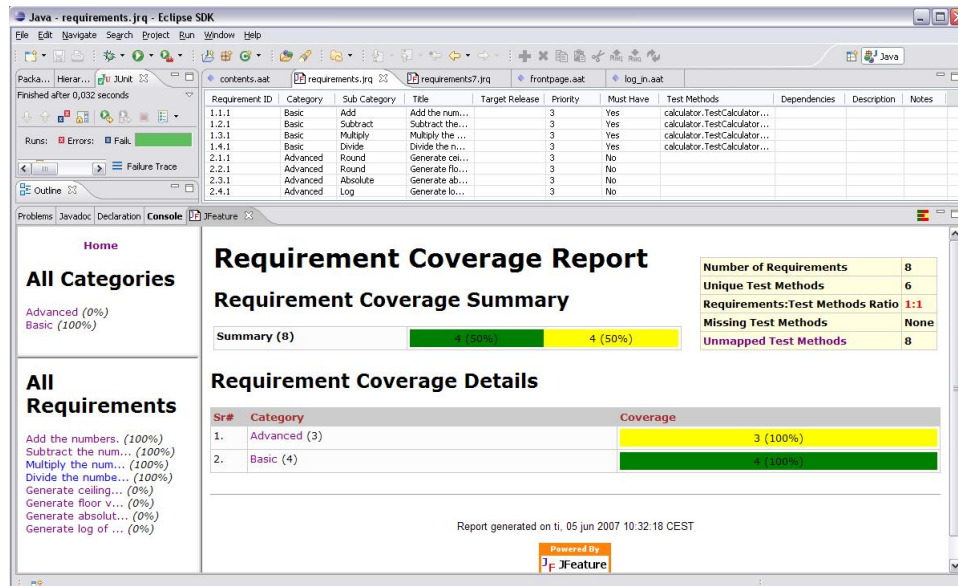


Figure 13.2: The Requirement Coverage Report

The requirement coverage report can be generated when the JUnit tests are run against the code, and the .jrj file is associated with the project containing the JUnit tests. The report provides the user with an overview over the requirement coverage for the main requirement categories specified in the .jrj file. When clicking on one of the main categories, the user will get an overview of the coverage with respect to the sub-categories in the .jrj file. When clicking on the sub-category, the user will get an overview over the titles belonging to that sub-category, and further, when clicking on the title, an overview over all belonging test methods and the coverage for each method will appear.

Part III
Own Contribution

The focus in this part is on the design and possible development of CubicProject. The first chapter describes a possible design solution with two possible platform solutions and in the end a discussion around the solutions with domain experts. The second chapter in this part describes the implementation of the chosen solution.

CHAPTER 14

CUBICPROJECT DESIGN

In this chapter we will present a solution which will fulfill the goals of this project. In section 14.1, we define the hierarchical structure of CubicProject, and in section 14.2, we present the user stories for the application, which tells us how the application should work and what possibilities it gives the developer. In the next sections, two platform solutions are presented, and the last section, section 14.6, contains a summary from the workshop where these solutions were discussed with project managers and developers.

14.1 Metaphors

The metaphors are chosen mainly with background in the agile methodologies discussed in chapter 6 and how CubicTest is designed and constructed. We believe that an important aspect is not to design CubicProject considering one agile method, but make the application useful regardless of which agile method it is intended to be used together with. In addition, CubicProject should easily include CubicTest, without requiring changes in CubicTest's design.

Elements

Projects, iterations, requirements, tasks, tests and status messages are all elements.

Sub elements

Sub elements are project elements, iteration elements, requirement elements, task elements, tests and error messages.

Project

A project consists of a set of project elements.

Iteration

An iteration is a project element and consists of a set of iteration elements. An iteration element is a project element, but a project element is not an iteration element.

Requirement

A requirement is an iteration element and consists of a set of requirement elements. A requirement element is an iteration element, but an iteration element is not necessarily a requirement element.

Task

A task is a requirement element and consists of a set of task elements. A task element is a requirement element, but a requirement element is not necessarily a task element.

Test

A test is a task element and consists of a status message.

Status Message

A status message is a task element. An error message is a type of status message.

Estimation Element

Project, iteration, requirement and task are estimation elements.

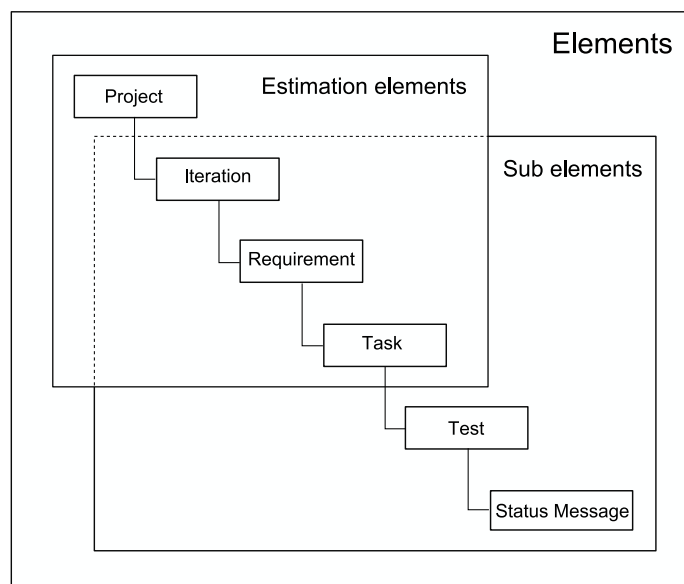


Figure 14.1: Metaphors

14.2 User Stories

User stories were introduced in section 6.1 on page 18. We have used user-stories because they describe the features of CubicProject in an intelligible and easy-to-understand way. The user stories presented below are sufficient for CubicProject to be realized and meet the goals of this project.

US1: Elements

A user should be able to create, read, update and delete elements and their belonging attributes.

US2: Sub Elements

A user should be able to connect sub elements to an element and get a view over which sub elements that are directly or indirectly connected to an element.

US3: Requirements

It should be possible to describe a requirement as a user-story, use case, or another format that is adjusted to the project or the requirement one wishes to describe, cf. several (agile) methodologies.

US4: Tests

It should be possible to add several sorts of tests (e.g. unit tests, acceptance tests, integration tests).

US5: Status

A user should be able to look at a project if a belonging project element, directly or indirectly connected, have an unsolved status message connected to it. It should be possible to get an overview over the status messages and to look at individual messages. This gives the user the project status at any given time.

US6: Responsible

It should be possible to connect a responsible person to a project and a project element.

US7: Estimation Element

It should be possible to estimate the size of every element that is an estimation element. It should also be possible to add a start date and an end date. It should be possible to observe the history of estimations done, to keep this history updated. Estimates should be given as minimum, maximum and most probable estimations. Elements can be estimated in hours or use case points (a relative size). All estimation elements should be updated based on status of run tests. When all tests belonging to an estimation element run correctly, one know that the element is implemented, assuming the tests are written correctly. Estimation elements should also contain how much time that is used on an element and how much time is needed to finish it.

US8: External Services

CubicProject should offer the opportunity to include external services in a project. The services should be able to create events in the project.

US9: History

A user should get a view over all events and changes done in the project, both internal events and events generated by external services. One example of history is when code has been checked in to the project.

US10: User

To take responsibility for a problem requires that users are defined in the system. One should get an overview over elements assigned to one self or other users.

US11: Template

A user should be able to store an element as a template for later reuse. The user should be able to adjust each element to let the element contain attributes a user needs. The attributes should also have optional editable names.

US12: Overview of Experience

A user should be able to store experience with all estimation elements and to get them later when estimating a similar element.

14.3 Domain Model

A domain model is a conceptual model of a system, providing a structural overview of the system by visualizing the entities of the system and their relationship. This is done here to document the key concepts of the system, as a supplement to the user stories and the metaphors.

To visualize the metaphors and user-stories, we have created a domain model, shown in figure 14.2. Our domain is *project management and traceability between tests and requirements*. The metaphors, which defines the main entities, and the user stories, which describes the features each entity shall offer, are together the requirements for the system.

The model shows the main entities described in section 14.1 and how they are interacting with each other, together with other classes and interfaces we believe are necessary. In the model we find the *EstimationElement* element, which is a generalization for *Project*, *Iteration*, *Requirement*, *Task*, *Test* and *Status*. These elements and their relationships are covered in US1 - US5, and US7. Every Estimation element should have a responsible, as stated in US6. This requires the *User* entity - that users are defined in the system, which is stated in US10. A project should be able to interact with *External Services*, as stated in US8. A user should be provided with an overview of the *Estimation Elements* history, that is, events and changes done in a *Project*. This is covered in US9. *EstimationElements* should be able to

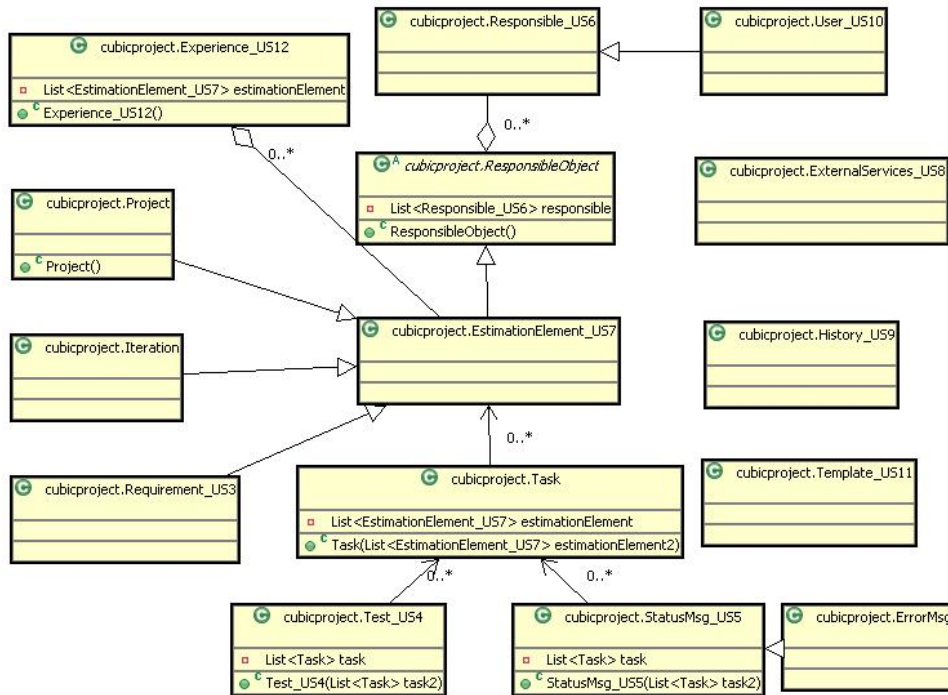


Figure 14.2: Domain Model

be stored as *Templates*, where the *User* can customize the stored element for later reuse. The *Template* feature is covered in US11. CubicProject should also be able to collect the *Experience* of *EstimationElements*, for later use when estimating similar elements. This is stated in US12.

We believe this model and the belonging user-stories, are a well designed foundation for our CubicProject solution. All details regarding functionality are not specified here, these details will appear if the solution is implemented.

14.4 CubicProject Solution Using Eclipse

This section describes how CubicProject can be realized as an application running on the Eclipse platform. The solution is partly based on the domain model and user-stories presented in the previous chapters, and partly on the storage discussion in chapter 10 and the platform discussion in chapter 11. From the user-stories in section 14.2 it is stated that events and changes done in projects shall be stored, including changes caused by external services. What we want to store is information, not files containing data. To use CVS or SVN with version control is a waste when the purpose is only to store plain information. Using CVS or SVN would make us have to collect the data into files before storing them, which doesn't seem convenient. The advantage with using a Database Management System (DBMS) presented in chapter 10.3, is among others that it is easy to set up, and have a special query language for altering the database. It is also easy to distribute the data if that is desirable. In addition, the installation and setup of a DBMS-solution can to a certain

degree be automated.

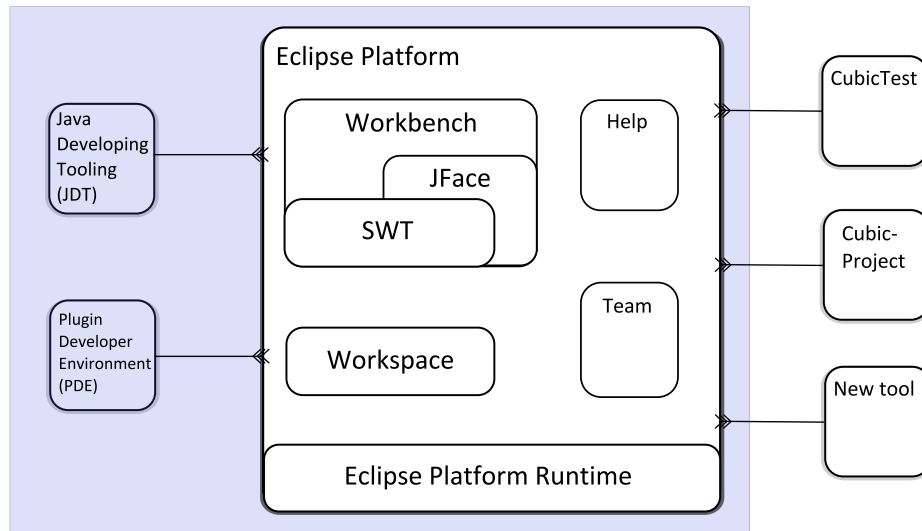


Figure 14.3: Eclipse Plugin Architecture

The aim of this thesis is to integrate CubicTest with a project management tool to realize traceability between tests modeled in CubicTest and requirements. As seen in chapter 7, CubicTest is developed as an Eclipse Plugin. From the study of the Eclipse Architecture in chapter 12, we learn that the Eclipse Platform is built up by plugins that provide extension points for external plugins to easily connect and provide features. Developing CubicProject as an Eclipse Plugin will let us benefit from the Eclipse framework, and let us easily integrate CubicTest into CubicProject.

In section 11.3 it is also stated that the Eclipse Platform will support the application with keyboard shortcuts, support for drag & drop functionality, and in the application we also have the opportunity to benefit from other Eclipse Plugins. In addition, the Eclipse IDE is widely used, and many developers and project managers are familiar with it. The biggest drawback is that non-technical users could have a hard time setting up the application, and all aspects regarding design and possible features are restricted to what the Eclipse Platform and plugins supporting additional features can provide.

With the solution presented in this section, the user can easily add requirements and assign them to different tests while modeling the information flow in CubicTest. After the code is implemented, when running the tests, the requirement coverage and possibly failing tests will automatically be updated in CubicProject, in addition to which tests succeeded.

14.5 CubicProject Solution using ESB

An alternative to a fully implemented Eclipse plugin is an ESB¹[24] solution. An ESB offers a solution where all kinds of services can communicate with each other,

¹Enterprise Service Bus

and works as a connectivity layer between the services. In this way, old existing systems or services can easily be integrated with each other and newer smaller services, instead of implementing a new large solution from scratch.

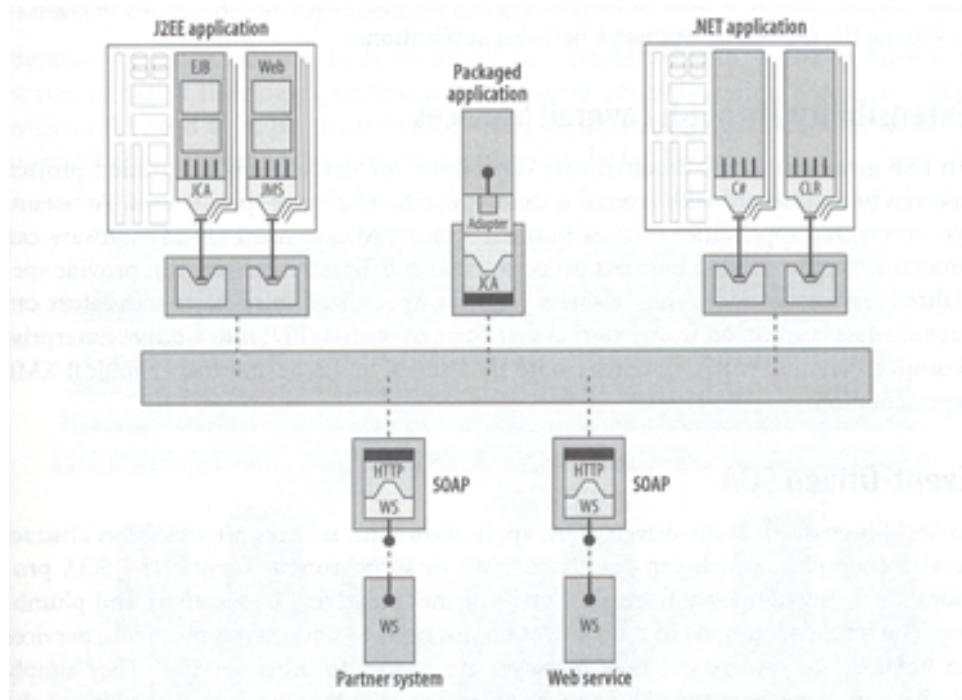


Figure 14.4: An ESB integrating a variety of disparate technologies [3]

ESB solutions use XML² as the standard communication language. It supports web-services³ standards and several Message Exchange Patterns⁴. The core has a construct that is both extensible and scalable and provide a standardized security model. The ESB is constructed to become as common as possible by using open communication standards, if such are available, to easily be able to integrate as many existing systems as possible.

An advantage is that the ESB solution can easily integrate (sub)systems that run in geographically different locations. It also increases flexibility and makes it easy to change part of the system when the requirements to the system change, because any application can become plugged into an ESB network using several connectivity options, and from that moment participate in sharing data with other applications, exposed as shared services across the bus. In addition, small individual projects can be built into a larger integration network in an incremental way. The project can be managed remotely from anywhere on the bus. The solution probably requires more system configuration than integration coding.

²eXtensible Markup Language

³<http://www.w3.org/2002/ws/>

⁴<http://www.w3.org/2002/ws/cg/2/07/meps.html>

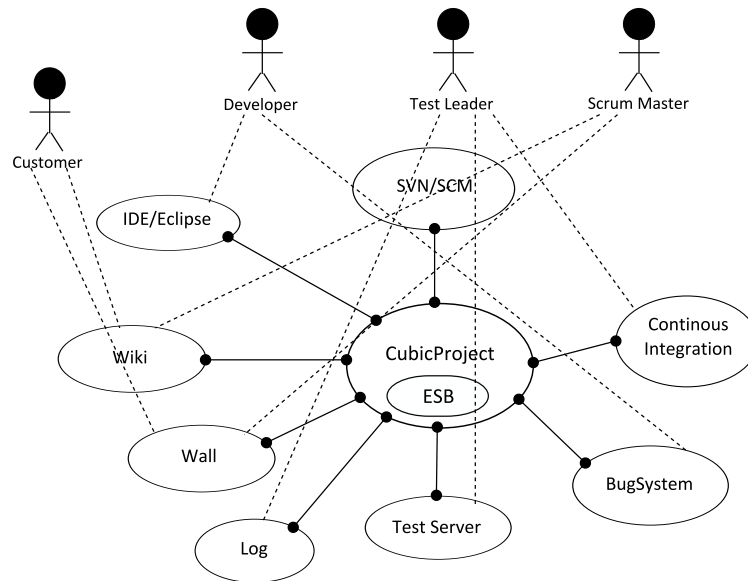


Figure 14.5: Enterprise Service Bus Solution

The figure below shows how we believe an ESB could connect existing services and applications to provide us with a powerful project management tool. This solution will provide us with the main target of this project, the traceability between tests and requirements, in addition to several other useful features related to project management. We believe this solution will assist all the stakeholders connected to a development project, since a stakeholder can easily access several features connected to the ESB, and also that more than one stakeholder can access one or more features at the same time.

For the customer, the most important features in our solution will be the Wiki and the wall. The wall originates from the physical wall where project managers attached their post-it notes with user stories or requirements. We have included the wall because the customer is familiar with it and use of the wall is widespread in agile software development projects. The Wiki will provide a presentation of the project progress, easily accessible for the customer and other interested parties.

The developers use a preferred IDE, e.g. Eclipse, to develop the software. They will also be able to connect the IDE with e.g. a bug system solution, to receive frequent accurate updates of errors and bugs in the code.

The test leader, if such one is present in the project, will be able to supervise the project and carry on his / her responsible through the project lifetime with help from a log, a test server and a continuous integration feature, all able to speak with each other in the same language. All these features will also, through the ESB, be able to receive information from e.g. the IDE and the bug system, process this information and pass new information on to e.g. the Wiki.

The Scrum master, or the project manager, if other management methods are used during the project, will also be interested into using the wall and the wiki to follow

the project progression. The scrum master can still be interested in gaining more insight in how the project develops, by for example following the log and receiving information from the test server. While all the features integrated with the ESB speak the same language, all information are accessible for the different features.

14.6 Focus Group

The solutions above were created on the basis of our view of what we believe the business needed. We are not experts in the domain and our assumptions about how software development works is a rather thin foundation to base the design of CubicProject on. To get an indication of whether we were heading in the right direction, we decided to discuss the solutions with project managers and developers that daily experience the issues we are trying to address.

Ideally we should have arranged several interviews with focus groups to create a broad data basis for answering our issues, but due to time and traveling constraints and the difficulties with available working hours for developers and project managers, we decided to attend an event called XpMeetUp, where project managers and developers get together and listen to people talking about subjects within agile methodologies, to have a focus group with some of the domain experts attending, after the lecture. In section 14.6.1 follows a summary from the focus group, highlighting the project managers and developers opinions. The issues below were only used as input to the discussion and to prevent the discussion from losing focus, hence several of the issues may not be answered directly in the summary.

The following issues where used as input to the discussion:

- Do project managers want tools that systematize project artifacts by predefined models, or is this something project managers want to decide themselves?
- What do project managers and developers expect from a project management tool?
- Do project managers have specific requirements regarding what features and functions a project management tool should provide?
- Do project managers have opinions about what platform a project management tool should be based on?
- Do project managers have experience with project management tools? Useful or not? Do advanced tools help the user?
- Do project managers have opinions on whether the focus should be on the project or on management?
- What features and functions do project managers and developers really want? What do they miss?
- Are there needs tied to specific roles in a project?

- Do project managers have opinions about traceability, from bug to test to requirement to project? How are tests related to requirements? Is the focus on test results or on fulfilled/not fulfilled requirements?
- Do project managers have opinions regarding our proposed solutions? Design? Features?

14.6.1 Summary

It is important to remember that there isn't a single unambiguous answer to all these questions. What features in a project management tool that is suitable for projects depends to a large degree on how the project is intended to be accomplished, the size of the project and the project team, and how the customer prioritize the project artifacts. When it comes to handling requirements in project management tools, it must be taken into consideration that requirements are described and specified differently - e.g. use cases versus user-stories. Some are concrete while others are loosely specified, and a suitable tool must be able to handle both. In addition, it must be possible to specify and change requirements gradually until tests are ready to run. In addition to projects' different needs, people involved in projects have different needs too, regarding focus, how they cooperate and habits when they work and plan the project. These things are important to remember; a tool which is custom-made for one type of projects, may become difficult to use within other types of projects.

An important problem for project management tools is the need to collect data and information from other systems. Some project management tools support import and/or export of data, but often in different formats, although the XML format is becoming increasingly common. This complicates integration with existing systems, and the work to modify and adjust them can quickly grow above the gain of integrating them. In addition to be able to pass information, the information must be structured in a way that both systems are able to recognize. This is the main issue regarding traceability between tests and requirements.

The participants were not impressed by our ESB solution, shown in figure 14.4. They don't agree that it is necessary to integrate all subsystems with each other. One reason for this is that it shouldn't be necessary for all subsystems to talk to each other and exchange information. With basis in Lean Software Development, section 6.3, it shouldn't be necessary for a tool to be able to keep track of bugs. If this need exists, it means that there are too much bugs, and the solution isn't to get a better overview over them. Too much bugs are an indication of something fundamentally wrong with the process being used and hence, the process must be refined instead.

What the ESB offers is an improvement compared to developing something from scratch, but it requires a lot of time to modify the existing systems, before they can talk to each other. Both the project managers and the developers are unsure if it is useful to use the required time to modify existing software for use in an ESB solution, when maybe less time can be used to develop something new. In addition, a project management tool should be simple to use, especially for project managers which often don't have the same technical knowledge as developers.

A project management tool must not necessarily be large and comprehensive, offering “everything for everybody”. Especially the project managers expressed their satisfaction with Microsoft Excel. Excel made them capable of easily adjust and customize the tool according to how each project was intended to be accomplished. In addition, Excel only contained the functions and features they needed, and nothing more. This they felt was an advantage, because their experience with other tools had shown them that the tools offered a lot of features they felt they really didn’t need, and that complicated the use of the tool. In fact, they actually preferred separate tools doing separate operations.

One common question customers often ask project managers is “How much delayed is the project?” and thereafter the project managers ask the developers or the test leaders “where is the delay, and why?”. When using Excel, the project manager has an overview over how much delayed the project is, based on oral or written reports from the developers and/or test leaders, or based on acceptance test results run by the customer or the project manager himself. The feedback will tell something about where in the project the delay is, on a high level, e.g. in which iteration or maybe in which user story etc. It doesn’t necessarily tell anything about why there is a delay.

If one could easily extract information about why there has occurred a delay at any time, one could put this information together and do a little Lean thinking. One of the most important principles in the Lean methodology is the elimination of waste. Waste is recognized as everything that doesn’t directly contribute to working code. If one easily could discover why delay occurs, one could look at the causes behind the “why”, target it as waste, and eliminate it.

Several of the project managers and developers that participated in the discussion were of the opinion that traceability between tests and requirements answers this question, because then, each requirement will be connected to one or several tests, and the project management tool will provide information to the project manager about test coverage for each requirement and possible failing tests. It is then easy to find out why the test fails and correct it. It would be an advantage if these test results could automatically update the status in the project manager’s tool. Then the feedback would be much more accurate and faster compared to a manual update.

The arguments against complexity are also used against our Eclipse solution running on top of a database. The project managers and developers don’t see the need for collecting all historical information in a database. When using CubicTest as a test tool they agree with using Eclipse as platform for the tool, but the only feature they feel the tool needs to provide, is the mapping between tests and requirements. To be able to generate a simple updated report from this mapping, they claim that will provide them with sufficient information about the project progression.

The fact that Microsoft Office products are popular as project management tools is confirmed in an article [25] by Pete Behrens in Trail Ridge Consulting which presents results from a survey regarding use of agile project management tools. The results in the article are based on over 500 survey responses from 39 countries, and says that for small companies (less than 100 employees), Microsoft Office products stand for half of the total tool usage in each company, by average, when it comes to

managing requirements, user stories and the product backlogs. For large companies, this percentage is even higher.

Regarding iteration planning and tracking, Office products, on the average, hold near 60% of the tool usage in the 190 small companies and just above 40% of the tool usage in the 335 large companies. As a tool for test case tracking, Office holds the largest percentage for the 190 small companies, just above 30% of the tool usage on the average, and for large companies, around 35% of the tool usage. While Office tools as a test case tracking tool are on the average most popular among the 190 small companies responding to the survey, traditional tools are slightly more popular (close to 40%) than Office tools on the average for the 335 large companies. It is as a defect tracking tool Office products fall behind both Agile tools, traditional tools and other tools on the average for the 190 small companies, and fall behind internal tools on the average for the 335 large companies that responded to the survey.

14.6.2 Conclusion

The discussion with the project managers and developers gave us valuable insight into how they work with project management. This especially counts for how they use tools to help them and which aspects they value when they choose tools to use. The conclusions from the discussion are summed up below.

- Project managers don't favor large complex tools that seek to integrate all features that could be useful in connection with project management. What the project managers want is a small tool, that is easy to customize, requires little effort to use, and only provides the project managers with features and functions the intend to use during the project.
- Our ESB solution isn't a solution the project managers or developers would use because it is too complex, offers a lot of features they don't feel they need, and will require too much effort to connect to their existing system. Thus, the system will not help the project in any way. They prefer separate tools doing separate operations.
- A feature project managers and developers often feel they miss is an automatic answer to the questions: "How much delayed is the project" and "where is the delay". Project managers and developers feels it would be an advantage to be able to answer those questions without the need for oral feedback from the project team, while performing project management.
- A solution the stakeholders feel would solve the problem regarding project delay information, is a tool that could easily map tests to requirements provided by the customer, and with a little effort keep the information regarding requirement coverage and possible delays automatically updated as the project proceeds.

CHAPTER 15

CUBICPROJECT SOLUTION

Unfortunately, none of the solutions in the previous chapter were adequate to develop, because none of them were tools the business felt they needed and wanted to use in their projects. One can argue that stakeholders should have been involved in the discussion earlier in the design process, to steer the design of CubicProject in a direction that could have given the stakeholders a tool they were willing to use. Unfortunately, this wasn't possible in this project, due to time constraints. Less time used on accomplishing the prestudy could have provided us with more time available for the design process, but we felt the prestudy had to be accomplished the way it were, to give us valuable insight in the context of agile development and project management, and to make sure we didn't tried to develop something already developed in the past.

A tool that provides some of the features and functionality stated in section 9.8 and wanted by project managers and developers in section 14.6, is JFeature (chapter 13). The reason why JFeature wasn't included in the prestudy, chapter 9, is that the tool isn't an adequate project management tool, only providing requirement coverage for unit tests, inside an IDE¹. Yet, taking a more closer look at the tool, we are convinced that JFeature integrated with CubicTest, very well can be developed into CubicProject as a prototype of the demanded functionality in section 14.6.

An integration between CubicTest and JFeature will be able to answer the problem definition in chapter 2, fulfill our research goal in section 4.1, in addition to cover the following aspects, stated in both section 9.8 and section 14.6:

- The tool will provide traceability between acceptance tests in CubicTest, and requirements imported into JFeature.
- The way JFeature organizes requirements, the tool will not favor any agile method in particular.
- JFeature is, as CubicTest, an Eclipse plugin, hence CubicProject will also become completely integrated in Eclipse.
- In addition to provide statistics regarding requirement coverage, the tool will also provide the user with test status for each of the tests.

The rest of this chapter describes the changes done to JFeature to integrate it with CubicTest, presented in chapter 7, [7], [8], [21]. Since the solution is based on

¹http://en.wikipedia.org/wiki/Integrated_Development_Environment

two existing systems, we will not present the design of each system explicitly, but only present the design and implementation needed to integrate the two tools as a proof-of-concept of traceability between tests and requirements. This integration will provide traceability between tests in CubicTest and requirements in JFeature, which fulfills the most wanted functionality identified in section 9.8, and also is missed by project managers and developers, as stated in 14.6.

15.1 Implementation

The package structure and architecture in both CubicTest and JFeature are kept as they are. We have however made some changes to classes and methods inside the two tools, which are presented in this section, to make the tools able to communicate with each other in a way that makes the integration convenient for the user to use. The affected packages are in CubicTest's Selenium Exporter; *selenium.runner*, *selenium.holders* and in JFeature; *jfeature.core* and *jfeature.eclipse.common*. The sourcecode for the changes in these packages can be found in appendix A.

The following packages and classes are directly involved in the integration between CubicTest's Selenium Exporter and JFeature. Figure 15.1 shows how they relate to each other.:

- `org.cubictest.exporters.selenium.runner`
`RunnerSetup.java`
- `org.cubictest.exporters.selenium.runner.holders`
`SeleniumHolder.java`
- `net.technobuff.jfeature.core`
`JFeatureController.java`
`RequirementCoverageManager.java`
`RequirementCoverageManagerImpl.java`
- `net.technobuff.jfeature.eclipse.common`
`JFeaturePlugin.java`
- `net.technobuff.jfeature.eclipse.listener`
`JFeatureTestRunListener.java`

15.1.1 selenium.runner

This package only contains the class that sets up and runs the CubicTest tests using Selenium, and is located in `RunnerSetup.java`. The changes made in this class is instantiation of a new `JFeatureTestRunListener()` object, which is used to call the `testStarted()` and `testRunEnded()` methods inside the `JFeatureTestRunListener` class. The code can be found in section A.1.

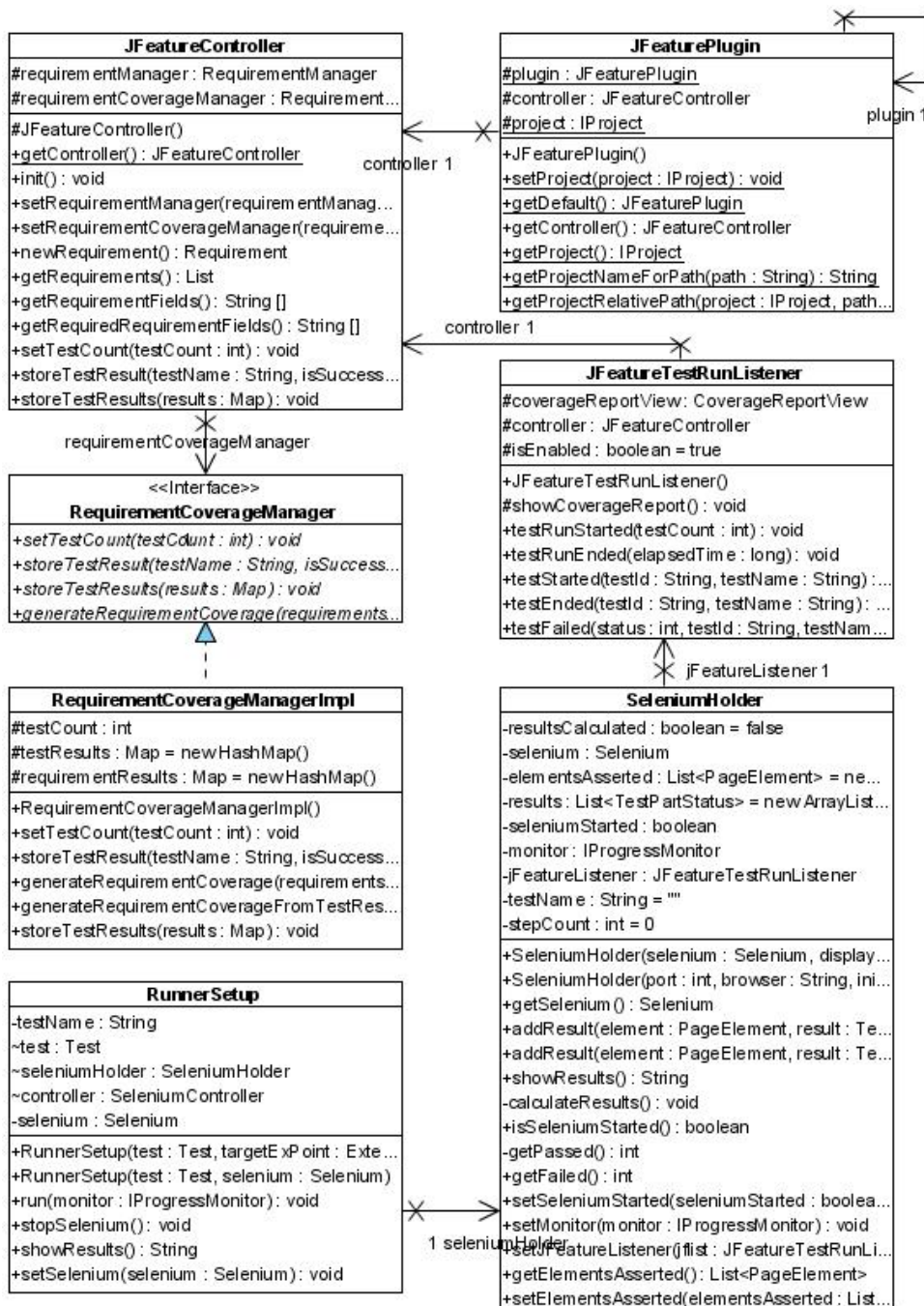


Figure 15.1: Class Diagram For The Solution

Originally, the `JFeatureTestRunListener` class was used to listen to running JUnit tests, but with the instantiation of a new `JFeatureTestRunListener()` object calling methods provided by the class, it now also listens to running Selenium tests from CubicTest. We have also added a creation of a `testName` string which creates a modified string of the running CubicTest test filename. This string, without the file path and the `.aat` extension, is then used as input parameter to the `testStarted()`

and `testRunEnded()` methods. Hence, the string works as an indicator for the running test.

15.1.2 selenium.runner.holders

The `addResult` method below is the method that gathers the results from each tested element in `CubicTest`, whether it is passed or failed. It is also responsible for showing the result immediately in the tested `.aat` file. Here, we have added a creation of a string `testStepName`, which holds the `testName` string and a `stepCount` integer.

Since Selenium doesn't distinguish between "User Action" elements, we have chosen to make a unique string, representing each element tested. Otherwise, `JFeature` will not count the correct number of elements tested, and thus the generated requirement coverage report will provide the user with incorrect information.

The `testName` contains the folder where the `.aat` file is located and the full name of the running `.aat` file. Including the folder name where the `.aat` file is stored, makes `JFeature` able to distinguish between several test `.aat` files with the same name. This could e.g. be different versions of the same test. A little drawback however, is that the folder also must be specified in the requirement `.jrq` file for each test. For each element tested in the `.aat` file, the `testStepName` string is concatenated with the name of the tested element and used as input parameter to either the `testEnded()` method, which marks the tested element as passed, or the `testFailed()` method, which marks the test of the element as failed. The code can be found in section A.2.

A new method `calculateResults()` is created since we no longer listen to `JUnit` test results. The method sets a test status for each element and increments either the total number of passed or failed tests. The `run()` method in `calculateResults()` are called by `get()` methods for the passed and failed steps. The method is shown in section A.3.

15.1.3 jfeature.core

The most important part of the `RequirementCoverageManagerImpl` class is the iteration over each test method where the test coverage is checked, and mapped on to the requirement categories specified in the `.jrq` file. The `while`-loop that performs the iteration had to be extended to work against both `JUnit` tests and `CubicTest` tests.

We created a new string `stub` which is a modified version of the `testMethod` string. We also created a new iterator `mIter`, that iterates over `testResults`. Then we cast the values iterated to a string, and match the string against the `stub` string mentioned above.

If there is a match, a new `testMethodCoverage` object is instantiated, and the test method iterated over is mapped onto the correct requirement category and sub-category. In addition, dependent on whether the result from the test is a success or a failure, the number of succeeded or failed items are incremented inside the `testMethodCoverage` instantiation. In the end of the loop, the test method coverage is added to the requirement coverage detail. If the `addCoverageDetail(testMethodCoverage)` isn't called by the `requirementCoverage` instantiation, the items tested

in the .aat file will not be visible as separate test methods in the generated requirement report. The changes made inside the class can be found in section A.4.

15.1.4 jfeature.eclipse.common

Since JFeaturePlugin originally only listens to JUnit runs, we had to create this method to make JFeature able to run against requirement .jrq files stored inside projects containing other test types than JUnit tests. While not removing anything, JFeature can still run against JUnit tests if desirable. The method that connects JFeature to the correct project, is shown in section A.5.

15.2 CubicProject UI and User Documentation

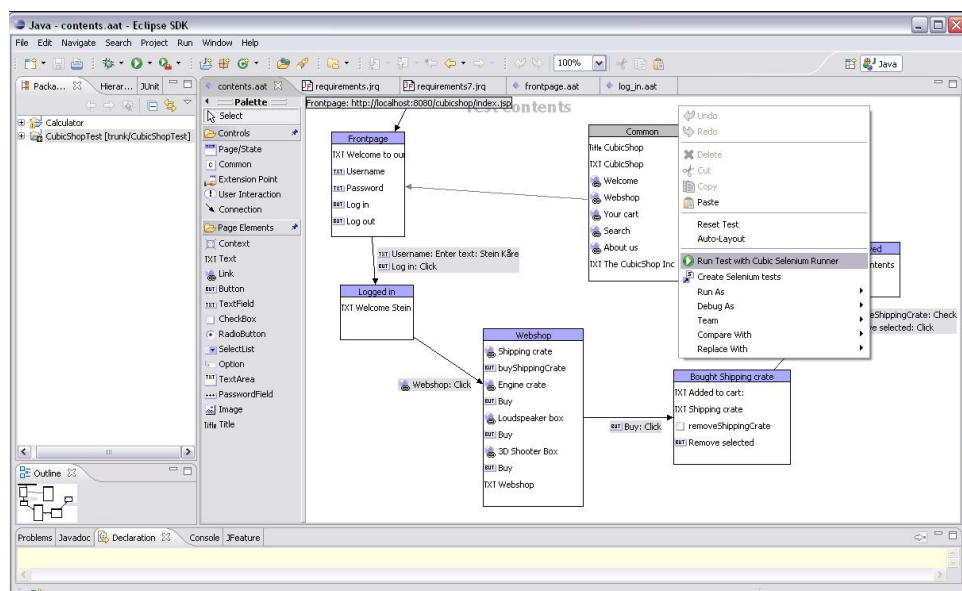


Figure 15.2: Running Tests In CubicTest Using The Selenium Exporter

To generate a requirement coverage report based on CubicTest tests run with the Selenium exporter, our modified versions of JFeature and CubicTestSeleniumExporter must be installed together with CubicTest. CubicTest can be checked out in Eclipse with SVN as a plugin project from <https://svn.openqa.org/svn/cubictest>, together with CubicRecorder [21], CubicTestWatirExporter [8] and the original CubicTestSeleniumExporter, which is the exporter that the CubicProject solution is based on.

The action visualized in figure 15.2 calls the `RunnerSetup.java`, described in section 15.1.1. Each element modeled in the test is then traversed and checked against the web page.

A status indicating whether the tested element succeeded or failed is returned and the element is marked accordingly - green for success and red for failure. When

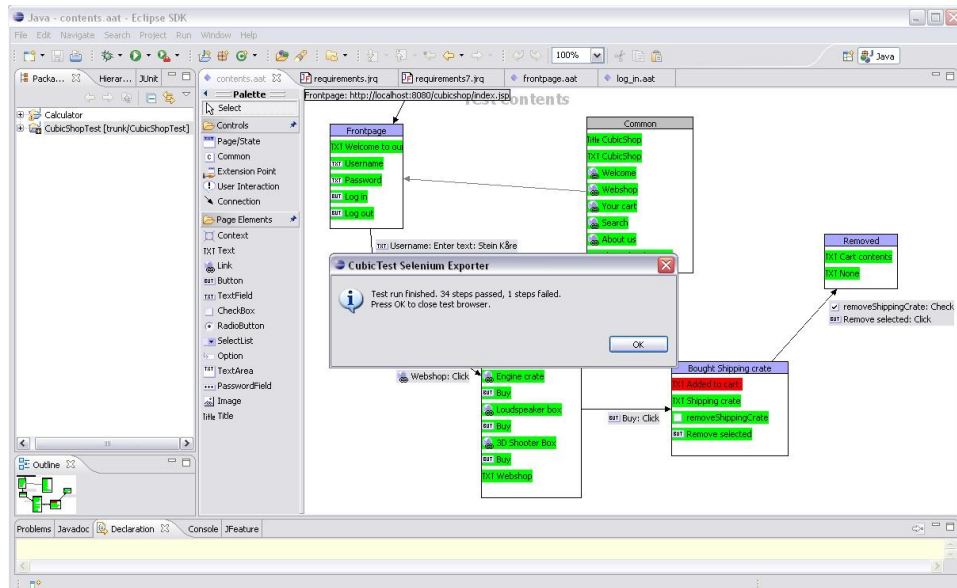


Figure 15.3: CubicTest Tests Finished

RunnerSetup.java is finished, a popup message is presented for the user, stating the number of succeeded and failed elements.

If the correct .jrj file is specified in the project properties under JFeature, and the correct test methods are mapped to the requirements inside the .jrj file, the requirement coverage report can be generated from the JFeature tab. The generated requirement coverage report is based on the results collected inside RequirementCoverageManagerImpl.java.

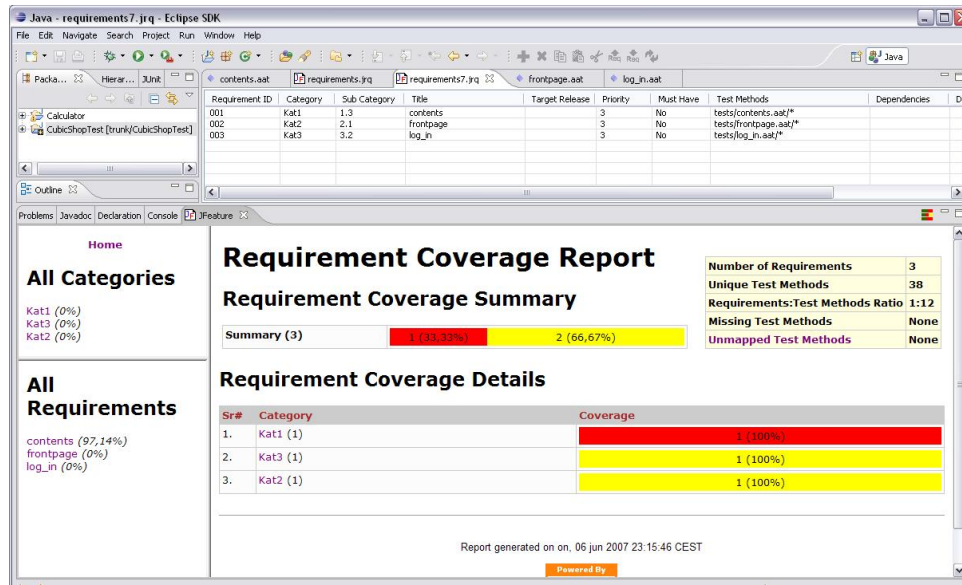


Figure 15.4: Generated Requirement Coverage Report

As can be seen on figure 15.4 on the bottom to the left, the 34 passed elements in figure 15.3 are summed up. Clicking on the link will give us a detailed report over the requirement coverage for the tested elements in the web page.

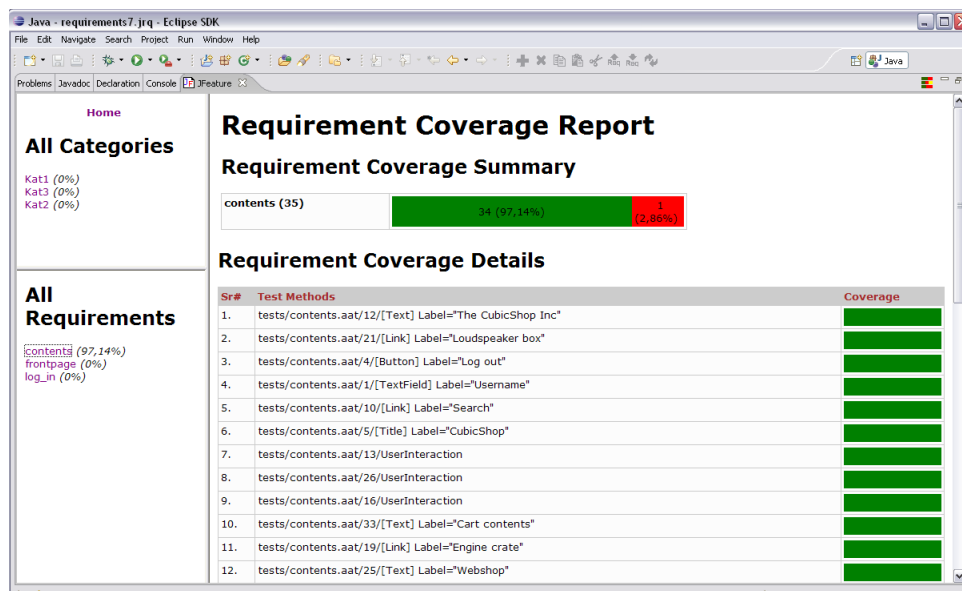


Figure 15.5: Detailed Requirement Coverage Report

Part IV

Evaluation

This part first presents the evaluation of the designed and implemented solutions, and of the project as a whole. Then we draw some conclusions based on the evaluation, followed by a chapter presenting possible further work with the project.

CHAPTER 16

DISCUSSION

The problem definition presented in chapter 2 was divided into two parts; studying existing open-source project management tools, and either integrate one of those tools with CubicTest or design a framework for a new tool, suitable for later integration. We will address these two parts in this chapter, in addition to own reflections regarding experience and the process as a whole.

16.1 The Prestudy

In chapter 6, we studied agile methodologies and several agile methods, and in chapter 9 we studied seven project management tools. We tried to identify the most well-known and widely used agile methods to focus on, since studying all existing methods would have taken too much time and provided us with too much information. Regarding the chosen tools, we tried to pick tools that seemed to focus on different aspects of project management, but kept the selection within open-source tools, since CubicTest are an open-source application. The narrow focus on project management was probably the reason why JFeature wasn't discovered here.

Some of the chosen tools were highlighted as project management tools for agile development, others designed for projects using traditional development methods. Some of the tools were web based, others had to be installed locally. In addition, some of the tools stored information locally in files, while the others used a DBMS solution. As we studied the tools, we became aware of several requirements that a suitable tool should fulfill. If one of the studied tools should be chosen, it had to contain these aspects to be appropriate for an integration with CubicTest. Since none of the tools contained all the functions, we decided to not go further with any of them.

16.2 The Design Process

When designing the solution proposal, we chose a DBMS solution as the most convenient solution for storing data. The main argument for this choice, was the need to store only plain information, not files containing data, distributed. This is the most convenient for the user when the information about requirement coverage, and other aspects within a project, were stored as plain information in a database. In this way, it will also be easier for the stakeholders to access the stored information,

and generate statistics based on it. The statistics will be especially important for the project manager when making project estimations in the future.

Several other solutions could have been considered here, for instance Distributed File Systems (DFSs). Compared to the rather small application which was the target in this project, using a DFS would add too much complexity to the system.

Choosing which platform solution to base our design on, wasn't a difficult decision to make. Since CubicTest is built on top of the Eclipse platform as a plugin, it would be a poor solution if CubicProject should run on a local-based or a web-based platform. Choosing to design CubicProject on top of a local-based or a web-based platform would evidently lead to major changes in CubicTest, changes which aren't wanted. In addition, redesigning CubicTest would draw much of the focus away from the original goals of the project - provide an integration between tests in CubicTest and requirements in a project management tool.

The solution we designed, we thought were complete and well suited for use in project management of agile development projects. Thus, we were quite surprised when the response from the focus group was that such a tool was neither appropriate nor desired among the project managers and developers that participated. The participants felt that neither the Eclipse solution designed from the bottom nor the ESB solution, were what they needed, even if both solutions contained the desired functionality. They felt the solutions were too complex and provided them with too much unwanted features and functionality. This would make the solutions difficult to use and navigate in. They were quite satisfied with the tools they had today, only missing a few features. One of the features they missed was to be able to trace tests back to the requirements provided by the customer.

We felt it was important to do a thorough study of existing tools and development methods to be sure that we emphasized the right aspects and didn't tried to design something that was already developed. On second thoughts, this part of the project might have taken too much time to complete, leaving too little time for design and implementation. In addition, the focus group session regarding what kind of tool the business felt they needed, should have started earlier. This way, we would have known earlier, if it was a product that would be used by the intended stakeholders. If none of the studied tools were found to be appropriate, we would have known in which direction to steer the design of a new tool before starting the design process. Hence, the time and effort used on designing something that the intended stakeholders felt were not appropriate for their work, could have been easily avoided, and instead used to design and develop a tool with only the desired features.

The question was then, refactor the Eclipse solution or come up with something brand new that would meet the needs of the project managers and developers. While considering this question, we came across the tool JFeature, chapter 13. Surprisingly, this tool seemed to be what the stakeholders from the workshop wanted, and what we had been trying to design. But even if our idea was realized by others, it didn't fulfill our goal, which was to provide traceability between requirements and tests modeled in CubicTest. JFeature only provides a connection between requirements and unit tests run against the JUnit test framework. Since JFeature is developed

as an Eclipse plugin, it would be a good foundation for a proof-of-concept of a tool that could integrate CubicTest to provide traceability between acceptance tests and requirements. Due to little time left at this point, we decided to prioritize this traceability, because the feedback during the focus group told us that this was most needed by the developers and project managers. We decided to integrate this tool with CubicTest and the integration itself turned out to be successful.

16.3 The CubicProject Tool

The solution with integrating JFeature and CubicTest is presented in chapter 15. The result of the integration is actually more a prototype than a proof-of-concept, since it hasn't been tested in use yet. However, since the tool provides a particular feature that the intended stakeholders have stated that they miss, we are confident that the tool is an improvement to project management with the use of CubicTest for acceptance testing. As a prototype, CubicProject has limitations and needs further development, and to be tested, before we can be sure that e.g. project managers actually gain something by using it. One feature that should have priority is to store the information regarding requirements and tests, and particularly requirement coverage, distributed. Due to the project's time constraints, we had to prioritize the work around realizing the traceability between the tests and the requirements.

CHAPTER 17

CONCLUSION

In this project, we have realized a prototype of an integration between CubicTest and an open-source project management tool, JFeature, that provides traceability between tests and requirements.

The research goal for this project was to provide traceability between tests in CubicTest and requirements in a chosen project management tool. The problem definition was divided into two parts, although the research goal for the project was the same, independent of the outcome of the two parts.

The first part was to study several state-of-the-art project management tools to find a suitable tool that could be integrated with CubicTest. To be able to look for and properly consider important aspects of the tools, we had to obtain information about the context of the project, which included agile methodologies and agile methods, project management and acceptance testing with CubicTest. We studied seven tools, but didn't find any of them appropriate to work with any further.

The second part was dependent of the first one. If we found a suitable tool, we were supposed to integrate the tool with CubicTest to fulfill our research goal. If a suitable tool wasn't found, a framework for a suitable tool that could fulfill our research goal should be designed, and if enough time was available, we should implement a prototype with the most important features. We designed two different solutions in parallel, but when we held a focus group with several project managers and developers to discuss the solutions, it turned out that none of the solutions would provide a tool that the intended stakeholders felt was useful in their work. As stated in the previous chapter, the focus group should ideally have been held earlier in the project.

At this time, we became aware of another tool called JFeature, which seemed to be able to fulfill our research goal. The reason why this tool wasn't discovered earlier, was that the tool focuses more on project development than project management. We integrated the tool with CubicTest, an integration that provides the desired traceability in section 9.8 and section 14.6.2.

We believe this solution to be a satisfactory answer to our problem definition in chapter 2 and fulfills our research goal presented in section 4.1, even if it is only a minimum fulfillment, and the traceability should ease both project management and increase the gain when using CubicTest for acceptance testing. The tool is only a prototype, and will need further development before it can be properly put into use.

Both CubicTest and JFeature are developed as Eclipse plugins, and hence, the integration of the tools, which we have called CubicProject, is well suited for further development, and will soon be added to its own repository at OpenQA.org¹, under the GNU General Public License version 2².

¹OpenQA states it self as the premier source for quality open source QA projects

²<http://www.gnu.org/licenses/gpl.txt>

CHAPTER 18

FURTHER WORK

As concluded in the previous chapter, CubicProject needs some additions and improvements before it can be properly used as a project management tool with automatic acceptance testing.

This thesis was aimed against one particular feature. The study of existing state-of-the-art tools presented in chapter 9 and the focus group with intended stakeholders, presented in section section 14.6, revealed at least one other desired feature, namely distributed storage of the requirement coverage information. To realize this, the distributed storage should also include the requirements and the tests. In this way, all project stakeholders will be able to access the information and with little effort be able to update the information that falls within each stakeholder's responsibility.

Related to distributed storage, it should be able to relate activities in the tool, such as modified requirement files and generated requirement coverage reports, to specified users. This way, it will be much easier to know who's responsible for changes done to tests or requirements.

Another needed improvement is to develop CubicProject as a separate plugin. As the prototype is today, one can only generate test coverage report from one single CubicTest test at the time. Developing CubicProject as a separate plugin will make the user able to run several CubicTest tests simultaneously and generate requirement coverage report from all the tests.

At last, the tool should be empirically tested in a real development project, to confirm that it is an improvement to acceptance testing in CubicTest and to project management as a whole.

Part V
Appendix

The appendix contains a chapter showing the code sections that integrates JFeature with CubicTest.

APPENDIX **A**

SOURCE CODE

A.1 selenium.runner

Listing A.1: RunnerSetup Excerpt

```
1
2 monitor.beginTask("Traversing the test model...", IProgressMonitor
   .UNKNOWN);
3
4     JFeaturePlugin.setProject(test.getProject());
5     JFeatureTestRunListener jflist = new
6         JFeatureTestRunListener();
7     jflist.testRunStarted(10);
8     seleniumHolder.setJFeatureListener(jflist);
9     String testName = test.getFile().getProjectRelativePath().
10        toPortableString();
11    testName = testName.replace("\\.aat$", "");
12    seleniumHolder.setTestName(testName);
13    jflist.testStarted("main", testName);
14    testWalker.convertTest(test, targetExPoint, seleniumHolder);
15    jflist.testRunEnded(10);
16
17    monitor.done();
```

A.2 selenium.runner.holders

Listing A.2: SeleniumHolder addResult Excerption

```
1
2 public void addResult(final PageElement element, TestPartStatus
   result) {
3     String testStepName = testName + "/" + stepCount + "/";
4     stepCount++;
5     handleUserCancel();
6     elementsAsserted.add(element);
7     results.add(result);
8     if(jFeatureListener != null) {
9         if(element != null) {
10            testStepName += element.toString().replaceAll("\\(..*?\\)",
               "");
11        } else {
12            testStepName += "UserInteraction";
13        }
14        System.out.println("testName: " + testStepName);
15        if(result.equals(TestPartStatus.PASS)) {
16            jFeatureListener.testEnded("testId", testStepName);
17        } else {
18            jFeatureListener.testFailed(0, "testId", testStepName, "")
               ;
19        }
20    }
21    //show result immediately in the GUI:
22    final TestPartStatus finalResult = result;
23    display.asyncExec(new Runnable() {
24        public void run() {
25            if(element != null)
26                element.setStatus(finalResult);
27        }
28    });
29 }
```

Listing A.3: SeleniumHolder calculateResults Excerpt

```
1
2 private void calculateResults() {
3     if(resultsCalculated) {
4         return;
5     }
6     resultsCalculated = true;
7     int i = 0;
8     for (PageElement element : elementsAsserted) {
9         if (element != null) {
10            final PageElement e = element;
11            final TestPartStatus res = results.get(i);
12            display.asyncExec(new Runnable() {
13                public void run() {
14                    e.setStatus(res);
15                }
16            });
17        }
18        if (results.get(i).equals(TestPartStatus.PASS)) {
19            passed++;
20        }
21        else {
22            failed++;
23        }
24        i++;
25    }
26 }
27 public String showResults() {
28     handleUserCancel();
29     return getPassed() + " steps passed, " + getFailed() + " steps
30         failed.";
```

A.3 jfeature.core

Listing A.4: RequirementCoverageManagerImpl Excerpt

```
1
2 if(testMethod.matches(".*\\*$")) {
3     String stub = "^" + testMethod.replaceAll("\\*$", "");
4     Iterator mIter = testResults.keySet().iterator();
5     isSuccess = new Boolean(true);
6     while(mIter.hasNext()) {
7         String m = (String) mIter.next();
8         if(m.matches(stub + ".*")) {
9             testMethodCoverage = new Coverage();
10            testMethodCoverage.setLabel(m);
11            testMethodCoverage.setItemType(TEST_METHOD_ITEM_TYPE);
12            if (requirement.categoryExists()) {
13                categoryTestcasesMap = (Map) categoriesUniqueTestcasesMap
14                    .get(category);
15                if (categoryTestcasesMap == null) {
16                    categoryTestcasesMap = new HashMap();
17                    categoriesUniqueTestcasesMap.put(category,
18                        categoryTestcasesMap);
19                }
20                categoryTestcasesMap.put(m, m);
21                if (requirement.subCategoryExists()) {
22                    subCategoryTestcasesMap = (Map)
23                        subCategoriesUniqueTestcasesMap.get(subCategoryId);
24                    if (subCategoryTestcasesMap == null) {
25                        subCategoryTestcasesMap = new HashMap();
26                        subCategoriesUniqueTestcasesMap.put(subCategoryId,
27                            subCategoryTestcasesMap);
28                    }
29                    subCategoryTestcasesMap.put(m, m);
30                }
31            }
32            uniqueTestcasesMap.put(m, m);
33            unmappedTestcasesMap.remove(m);
34            Boolean mSuccess = (Boolean) testResults.get(m);
35            isSuccess = new Boolean(isSuccess.booleanValue() && mSuccess
36                .booleanValue());
37            if(mSuccess != null && mSuccess.booleanValue()) {
38                numberOfSuccessItems++;
39                testMethodCoverage.setNumberOfSuccessItems(
40                    testMethodCoverage.getNumberOfSuccessItems() + 1);
41            } else {
42                numberOfFailureItems++;
43                testMethodCoverage.setNumberOfFailureItems(
44                    testMethodCoverage.getNumberOfFailureItems() + 1);
45            }
46        }
47        // Add test method coverage to the requirement coverage
48        // detail
49        requirementCoverage.addCoverageDetail(testMethodCoverage);
50    }
51 }
52 if(isSuccess.booleanValue()) {
53     numberOfSummarySuccessItems++;
54 }
```



```
46     } else {  
47         numberOfSummaryFailureItems++;  
48     }  
49 }
```

A.4 jfeature.eclipse.common

Listing A.5: JFeaturePlugin setProject Excerpt

```
1  
2 public static void setProject(IProject project) {  
3     JFeaturePlugin.project = project;  
4 }
```

BIBLIOGRAPHY

- [1] Kent Beck and Cynthia Andres. *Extreme Programming Explained - Second Edition*. Pearson Education, Inc., 2004.
- [2] Ken Schwaber. *Agile Project Management With Scrum*. Microsoft Press, 2004.
- [3] David A. Chappel. *Enterprise Service Bus*. O'Reilly, 2004.
- [4] cromatic. *Extreme Programming - Pocket Guide*. O'Reilly, 2003.
- [5] Mary Poppendieck and Tom Poppendieck. *Lean Software Development - An Agile Toolkit*. Addison-Wesley, 2003.
- [6] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice-Hall, 2002.
- [7] Trond Marius Øvstetun and Stein Kåre Skyttern. Autat - automatic acceptance testing of web applications, 2005.
- [8] Kjersti Loe and Stine Lill Notto Olsen. Automatisert testing av dynamisk html, 2006.
- [9] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. June 9-11, 2003. Available at http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1231428.
- [10] Martin Fowler and Jim Highsmith. The agile manifesto. Available at <http://www.ddj.com/dept/architect/184414755>, Year = July 16, 2001.
- [11] Pascal Van Cauwenberghe. Refactoring or upfront design? 2003. Available at http://www.nayima.be/html/refactoring_or_upfront_xp2001.pdf.
- [12] Craig Larman. *Agile & Iterative Development - A Manager's Guide*. Pearson Education, Inc., 2004.
- [13] Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. January 1, 1986. Available at <http://apl-n-richmond.pbwiki.com/f/New%20New%20Prod%20Devel%20Game.pdf>.

- [14] Winston W. Royce. Managing the development of large software systems. 1970. Available at <http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>.
- [15] Dr. Jeff Sutherland. Agile development, lessons learned from the first scrum. October, 2004. Available at <http://jeffsutherland.com/scrums/FirstScrum2004.pdf>.
- [16] Controlled-chaos software development. Available at <http://www.controlchaos.com/download/Controlled-Chaos%20Software%20Development.pdf>.
- [17] Crystal methodologies. Available at http://utopia.csis.pace.edu/dps/2007/amannette-wright/dps/DCS801_821/Crystal_Methodologies_summary.doc.
- [18] Alistair Cockburn. *Crystal Clear - A Human-Powered Methodology for Small Teams*. Pearson Education, Inc., 2005.
- [19] Software project management: Methodologies & techniques. Available at <http://paul.luon.net/essays/SEP-essay-final.pdf>, September 17, 2004.
- [20] Elena Kalitina and Jon Reinert Myhre. Cubicstest usability, 2006.
- [21] Erlend Halvorsen. Automated acceptance testing of web applications with cubicstest, 2007.
- [22] Ben Collins-Sussman, Brian W Fitzpatrick, and C. Michael Pialtio. *Version Control with Subversion: For Subversion 1.4: (Compiled from r2769)*. TBA, 2007.
- [23] Eclipse platform technical overview. Available at <http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf>.
- [24] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The enterprise service bus: Making service-oriented architecture real. *Service-Oriented Architecture - Volume 44, Number 4, 2005*, October 24, 2005. Available at <http://www.research.ibm.com/journal/sj/444/schmidt.html>.
- [25] Pete Behrens. Agile project management (apm) tooling survey results. December, 2006. Available at <http://www.trailridgeconsulting.com/files/2006AgileToolingSurveyResults.pdf>.