NTNU
Norwegian University of
Science and Technology

Joining in Apache Derby: Removing the Obstacles

Henrik Holum
Svein Erik Reknes Løvland

Master of Science in Computer Science
Submission date: June 2008
Supervisor: Maria Letizia Jaccheri, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Apache Derby is an open source project started in 2004. At the time of writing there are 21 committers, 11 members of the Project Management Committee and 4 commercial companies working actively on the project, with Sun Microsystems and IBM as the major contributors. The community is very active, with up to several hundred mails on the mailing lists every day.

Joining means becoming an active developer in the Apache Derby project when starting with no previous knowledge about it.

Challenges will be to get an overview of the code, find where it is possible for a newcomer to contribute, identify obstacles and find suggestions to remove these.

The students will this semester look at Joining in Apache Derby. They will use Canonical Action Research as their research method when doing this.

The result will be a guideline on how to remove the obstacles which are identified.

The results can be used by others projects to lower their contribution barrier, specially other Apache projects.

Assignment given: 21. January 2008
Supervisor: Maria Letizia Jaccheri, IDI

Abstract

Over the last decade, the amount of commercial interest in Open Source has been growing rapidly. This has led to commercially driven Open Source projects. Those projects have problems keeping their newcomers and needs ways to ease the joining process. Therefore we ask these research questions:

- RQ1: Which obstacles are encountered by Newcomers to Apache Derby when Joining?
- RQ2: What can be done to ease the Joining process?

There has been very little research on what the OSS projects can do in this area. As a consequence it is hard to find good reliable theory to cross-reference this research. If the research is successful, it can contribute to the literature on joining in OSS projects. This literature will then contain all obstacles encountered by newcomers to OSS projects and ways to mitigate these.

In this master's thesis Canonical Action Research was used to study the Open Source project Apache Derby. Canonical Action Research is a qualitative research method where the researchers enters the environment they are researching to extract the data needed.

We have three contributions in this thesis. The first contribution is a list of obstacles in the joining process of Apache Derby. The second contribution is suggestions on how a project can mitigate the contribution barriers we found. The third contribution is a refined version of CAR to use when studying Open Source Software Development.

The list of obstacles is a contribution specific to the Apache Derby project, and it is very unlikely that other non Apache projects will benefit from it. Our suggestions on how a project can mitigate contribution barriers are potentially generalizable. Different projects have different structures, and some of the contribution barriers might therefore not apply to them all. The refined CAR model is general for all research on OSS projects. This is the result we think can have the biggest impact on the research community if proven successful.

Preface

This thesis is the result of a semester's work, from January to June 2008. It is part of the subject TDT4900, Program and Information Systems, Master's thesis. This subject is the final step of our Masters degree at the Norwegian University of Science and Technology(NTNU), Department of Computer and Information Science(IDI).

We have used Canonical Action Research to investigate how to ease the Joining process in an Open Source community, namely Apache Derby. This was a continuation of our previous work where we looked at testing in an Open Source community. We have had large quantum of literature to study. We were familiar with the concepts of Canonical Action Research after using it last semester, but we had to spend a lot of time to ensure the correctness of our research this semester as well. This has led us to be able to use a real research method to investigate interesting research questions and reflect on the outcome.

We would like to thank Maria Letizia Jaccheri for her guidance throughout our work on this report. We would also like to thank IDI for providing us with a working space and the hardware needed for this project. Henrik would like to thank his girlfriend for putting up with his early mornings over the last months of this project. Svein Erik would like to thank Carl-Fredrik Sørensen for good advice. Last but not least we would like to thanks all our fellow students, specially those who shared the office space with us at Gribb, that have endured our endless discussions on the subjects of this thesis.

Trondheim, 13. June 2008

Henrik Holum
holum@stud.ntnu.no

Svein Erik Reknes Løvland
sveinelo@stud.ntnu.no

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter is an introduction to this thesis. The context is explained, the research method is presented as well as our research questions and contributions. Finally the report outline is given.

1.1 Context

This report was written as part of the subject TDT4900, Program and Information Systems, Masters Thesis. It is a continuation of our work from last semester, where we aimed to shed light on testing in an Open Source Software Development (OSSD) setting. These projects have a lot of similarities and we have chosen to use our old pre-study as the foundation for the pre-study in this project. We did however have a much more thorough understanding of Open Source and Canonical Action Research this semester, giving us the opportunity to dive much deeper into more specialized literature, enabling us to write a much more in-depth State-of-the-art. We are two students, with a general interest in Open Source, working on this as a semesters work. The project was supervised by professor Maria Letizia Jaccheri who is working at the Department of Computer and Information Science (IDI), NTNU. She accepted our problem description, which builds upon our problem description from last semester.

A short description of our previous assignment is: **Studying Open Source Software with Action Research**. The reason we chose this project is because OSS have been rapidly increasing in importance over the last decade. This fact, combined with the characteristics of Canonical Action Research, made this a very interesting theme for us as students. We chose to work with Apache Derby, which is a mature and stable OSS project.

1.2 Canonical Action Research

"CAR aims to address organizational problems while at the same time contributing to scholarly knowledge" [Davison et al., 2004]. Canonical Action Research (CAR) gave us a unique chance to get involved in a project, improve our own technical skills and contribute to an Open Source Software (OSS) project while at the same time do research and extract knowledge. In this thesis we used the five principles outlined in [Davison et al., 2004] as a basis to structure our work.

1.3 Research Questions

Below you can see the research questions we answer in this thesis.

- RQ1: Which obstacles are encountered by Newcomers to Apache Derby when Joining?
- RQ2: What can be done to ease the Joining process?

Apache Derby is a mature project. The obstacles found while working with RQ1 will therefore be of interest to other Open Source projects. There exists commercial interest in getting new developers. By answering RQ2 this thesis will increase the chance of Newcomers staying with a project and thereby adding value to it.

1.4 Contributions

As this thesis will show, we have answered the RQs listed above. By doing this we have increased knowledge concerning :

- Obstacles in the Joining process of Apache Derby.
- How a project can help mitigate its Contribution Barriers.
- The use of CAR as a research method in OSSD projects.

1.5 Report Outline

The structure of this report is as follows:

- Chapter 1: Introduction
  Project context, project objectives and report structure.

- Chapter 2: Research Method
  Presentation of Canonical Action Research.
- Chapter 3: Open Source Software
  An introduction to Open Source.
- Chapter 4: Joining
  This is our State-of-the-art chapter. Definitions needed for the report is presented.
- Chapter 5:Analysis of the Situation
  Analysis of the situation in Apache Derby with regards to our State-of-the-art chapter. Here we also present our Research Questions.
- Chapter 6: Iterations
  Our iterations. Includes a summary of our previous work.
- Chapter 7: Discussion
  Here we discuss our research and findings. We will also criticise our work.
- Chapter 8: Conclusions and Further Work
  Here our conclusions and further work are presented.

Appendices:

- Appendix A: Abbreviations
- Appendix B: JUnit URLs
- Appendix C: Researchers-Client Agreement

Part I

Prestudy

Chapter 2

Research method

We will in this project use Canonical Action Research(CAR), a derivative of Action Research(AR) when looking into Joining in the Open Source project Apache Derby. In this chapter we will present the research method CAR.

2.1 History of Action Research

Lewin (1951) get credited for developing Action Research (AR) at the Research Centre for Group Dynamics (University of Michigan) in order to study social psychology within the framework of field theory. Independently of the work of Lewin, a group of researchers at the Tavistock Clinic (later renamed Tavistock Institute)) developed a similar approach [Trist, 1976]. After World War II there were thousands of cases of psychological and social disorders caused by battle-fields and prisoner-of-war camps. The Tavistock Institute dealt with these kinds of problems. These psychological syndromes had not been identified in such a large population of patients before, and throughout the studies you could see that each case was a bit 'different' "Hence, the idea of social action arose. Scientists intervened in each experimental case by changing some aspect of the patients' being or surroundings. Since scientist and therapist were one, the scientists were participants in their own research"[Baskerville and Wood-Harper, 1996]. The researcher recorded the results in each case, and let others access this data. In this manner, a body of knowledge was developed about successful therapy for the illnesses (cf, [Rapoport, 1970]).

Lewin organised this research method into six phases, opposed to the five phases we use in CAR today. The original six phases were (1)analysis, (2)fact-finding, (3)conceptualization, (4)planning, (5)implementation of action, and (6)evaluation. These steps are very similar to the five steps we today use in CAR. From the original model to todays model, there has been alot of modifications. In figure 2.1 you can see a few different models presented by, and one created by, [Iversen et al., 2004]. The principle of the Cyclical Process Model (CPM) is used

	(Susman and Evered 1978)	(Checkland 1991)	(McKay and Marshall 2001)	Our CPR-Based Process
Initiating	Establish the client-system infrastructure	1. Enter problem situation	1. Identify: problem and research theme 2. Reconnaissance: problem context and research literature 3. Plan and design: problem solving and research questions	1. Appreciate problem situation 2. Study literature 3. Select risk approach
Iterating	1. Diagnosing 2. Action planning 3. Action taking 4. Evaluating 5. Specifying learning	2. Establish roles 3. Declare framework (F) and methodology (M) 4. Take part in change process 5. Rethink 2-4	4. Action steps 5. Implement 6. Monitor: problem solving and research 7. Evaluate in terms of problem alleviation and research questions 8. Amend plan based on 7	4. Develop risk framework 5. Design risk process 6. Apply approach 7. Evaluate experiences
Closing		6. Exit 7. Reflect on experience and record learning in relation to F, M, and problem situation	9. Exit, if: problems alleviated and research questions resolved	8. Exit 9. Assess usefulness 10. Elicit research results

Figure 2.1: Overview of Action Research Process [Iversen et al., 2004].

by all models, and is the cornerstone of AR. The main reason for the different models is the modifiability of the AR model. Two projects are never the same, and each project will have some distinct attributes that requires unique modifications to the model. This is one of the strengths of AR, the possibility to modify the model to suite the project. We follow the principles of CAR in our research project, and as will be seen, this is yet another modification of AR.

2.2 Canonical Action Research

There are a lot of papers attacking the use of AR in computer science. "Action research (AR) is not without its critics, and those who reject some of the paradigmatic assumptions embodied in AR maintain that AR is little more than consultancy, that it is impossible to establish causal relationships, that it is difficult to generalize from AR studies, that there is a risk of researcher bias, and that generally speaking, it lacks some of the key qualities that are normally associated with rigorous research." [McKay and Marshall, 2001]. These views are supported by [Avison et al., 1999]. An answer to this critique was written by [Davison et al., 2004]. Here he presents five principles to guide action researchers to ensure rigor and relevance in their research, and help readers of AR papers decide the quality of the articles. We follow these principles in our research. An outline of the principles follows.

2.2.1 Principle of the Researchers-Client Agreement

A Researchers-Client Agreement (RCA) is meant to give the client an understanding of how the research method works. The client should get an understanding of what might be expected and agree that CAR is an appropriate method for studying the subject. In the agreement responsibility of the researcher and client organization, objectives and evaluations measures, and data collection and analysis methods should be specified explicitly. This will help the client organization understand what is to take place, define in what way the researcher collect data and elaborate on how the experience will help improve the current situation.

2.2.2 Principle of the Cyclical Process Model

Cyclical Process Model (CPM) helps the researchers structure the task at hand. Each stage will produce an output which will be used as an input to the next stage. The stages in chronological order can be seen in figure 2.2. In table 2.1 you can see a description of the different stages in CPM. This model is described in detail by [Susmanand and Evered, 1978].

In the reflection stage we check if we have enough data to answer the research questions. If we do, then we can exit the project.

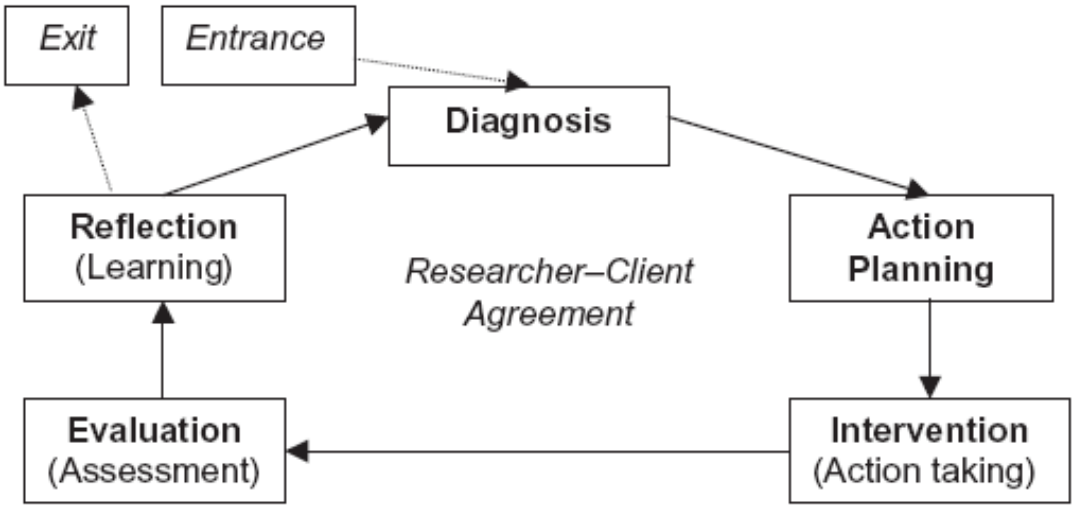

Figure 2.2: The Cyclic Process Model.

Stages	Description
Diagnosis	First a problem is addressed, then the situation is compared to existing theory. The client and the research community should agree that the problem addressed is of interest.
Action Planning	A plan is made according to the diagnosis. It addresses the problems detected in diagnosis.
Intervention	Perform the planned actions and collect data.
Evaluation	Analyze the collected data and the actions taken.
Reflection	Reflect on the results from the evaluation.

Table 2.1: Summary of the Cyclical Process Model.

2.2.3 Principle of Theory

Action Research without theory is in danger to become action learning. You have to tie your actions to knowledge, often academic papers, to help verify the validity of your research. Without theory there is no point in doing CAR. Theory makes sure the investigation is of interest for researchers and the client. It helps focus the research and improve the intervention stage so that it has a purpose and does not become pure consultancy. Theory is what separates action research from consulting.

2.2.4 Principle of Change Through Action

AR want to change the diagnosed problem to the better. If there is no change, then a new problem might have to be re-diagnosed. Action Research seeks to improve the current diagnosed situation by action taking in the environment. If there is no change in the diagnosed problem then this might be because the issue was not a problem after all, or the researcher might need another cycle in the CPM.

2.2.5 Principle of Learning Through Reflection

An action researcher has multiple stakeholders, with the research community and the client as the most important. The client wants an improvement in the situation while the research community want new knowledge. This is achieved by learning through reflection. It is vital to choose a subject to study which is of interest for both stakeholders. New knowledge might be to refine current methods. It is important to document the learning when it is found, so that the research will not be biased by retrospective reporting. This means that knowledge tends to change over time, and if you do not document it when you learn it you might not get the same result when you later sit down to recollect the situation.

Chapter 3

Open Source Software

This chapter gives a short introduction to the history of Open Source Software (OSS), an overview of the development process, an introduction to licensing and an overview of recent development. A specific look at the Apache Foundation can be found at the end of this chapter.

3.1 History of OSS

The people working with computers in the early years were either connected to the academic community or military. Sharing knowledge and findings are common among the people in academia and they shared their code and ideas. In the late sixties ARPANET was created, which is the predecessor of the Internet as we know it today. Previously Open Source hacker cultures had evolved at the different universities throughout the USA and with the ARPANET this culture really started to thrive [Raymond, 2001a]. They now had an electronic highway to exchange information, research and a now famous slang dictionary (Jargon File[1]), which helped build a common hacker culture.

Commercial companies started to protect their investments by distributing their software as binary packages without making the code available to the public. This annoyed some people, especially hackers and specifically Richard Stallman who initiated the GNU (GNU's Not Unix) project and started the Free Software Foundation[2] in 1985 [Beard and Kim, 2007]. FSF was founded with the ideology that software should be free for the user. In 1989 Stallman wrote the GNU General Public License (GPL) which is the most common OSS license in use today.

The term free in Free Software is ambiguous. It is often misinterpreted as free of charge. What it really means is stated in The Free Software Definition[3](see

[1] http://www.catb.org/jargon/

[2] www.fsf.org

[3] http://www.gnu.org/philosophy/free-sw.html

freedom 0	The freedom to run the program, for any purpose.
freedom 1	The freedom to study how the program works, and adapt it to your needs. Access to the source code is a precondition for this.
freedom 2	The freedom to redistribute copies so you can help your neighbor.
freedom 3	The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this.

Table 3.1: The Free Software Definition.

Table 3.1). Eric S. Raymond et al. suggested to use the term Open Source Software to Netscape in 1998. They wanted to; remove the misconception of the term free; make business open up their source; and understand the advantages the OSS development model gives. Netscape was the first to open their source from previously closed source with the Mozilla project. Lately the abbreviation FLOSS (Free/Libre/Open-Source Software) has been suggested.

The kernel Linux is a success story which have astound many people. It was started by Linus Torvalds, a finish student, in 1991. Together with the utilities and libraries from the GNU project it forms the GNU/Linux operating system which has been widely adopted [Wheeler, 2007]. Linux was seemingly randomly hacked together by many people, but rapidly evolved into a full featured Unix like system. Some of the well known successful OSS projects are Apache (httpd), Mozilla Firefox and OpenOffice.

3.2 Open Source Software Development Process

Research on Open Source Software Development (OSSD) have increased the in last years. [Østerlie and Jaccheri, 2007] argues that the literature describes OSSD as a homogeneous phenomenon and it has an implicit view of OSSD as different from Closed Source Software (CSS) development. This is a potential bias in most of the research on OSSD. [Gacek and Arief, 2004] notes that the only common characteristic of OSSD is that software product is released under a license compliant with the Open Source Definition [4]. A single generic OSSD process is therefore difficult to describe.

A description of OSSD processes can be found in [Raymond, 2001b]. Two ways of developing software are presented: the cathedral and the bazaar. The

[4] http://www.opensource.org/

development process of Linux is described as bazaar like, and it is this development style which is commonly thought of as the OSSD process. In bazaar like development many people seemingly randomly hacks the code together and there is little or no control of who works on what problem. The result is that more than one solution to a problem are made and can be chosen from. The project owner of Linux, Linus Torvalds, could choose the solution he though was most appropriate. This is opposed to the cathedral development style where problems are assigned and the assignee comes up with a solutions and no working alternative is made. Frequent releases of beta versions and rapid feedback from a large source of beta testers and co-developers is regarded as one of the strength of OSSD. It is also claimed that: "Given enough eyeballs, all bugs are shallow" (Linus's Law) [Raymond, 2001b]. If a project has a large enough user-base then all bugs will be reported, and eventually fixed.

Most OSS projects are organized as meritocracies. The commit regime has gained their status by showing their skills and gaining trusts from their co-developers. If the commit regime fails, the other developers are free to fork the project[5]. [Mockus et al., 2002] found that Open Source developments will have a core of developers who control the code base, and will create approximately 80% or more of the new functionality. If this core group uses only informal ad hoc means of coordinating their work, the group will be no larger than 10 to 15 people.

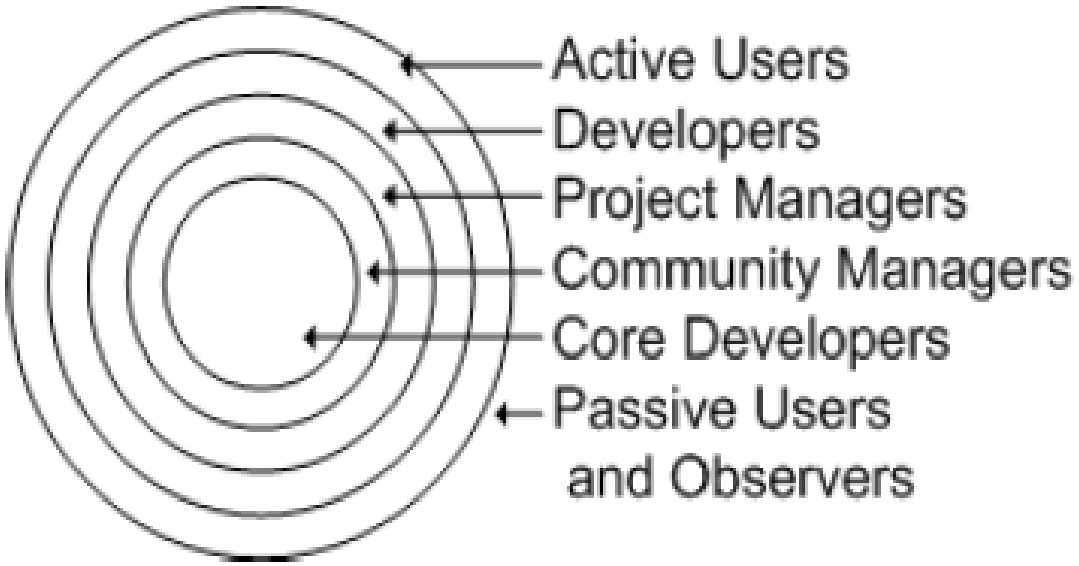

Figure 3.1: OSS Onion Model.

[5] "In software engineering, a project fork happens when developers take a copy of source code from one software package and start independent development on it, creating a distinct piece of software." - Wikipedia.org

The organization of an OSS community can be viewed as an onion-model (see Figure 3.1) with the core developers in the center and the users in the outer rim. [Jensen and Scacchi, 2007] looks into role migration and advancement processes in OSSD projects and found the onion-model inadequate. Role migration can be different from project to project, but climbing up a step in the community is almost always done in a meritocratic fashion.

3.3 Licensing

"The OSI (Open Source Initiative) are the stewards of the Open Source Definition (OSD) and the community-recognized body for reviewing and approving licenses as OSD-conformant"[6]. To call a software Open Source it must come with a license which is approved by this group. The GNU GPL license has been criticized for being viral and "executives from proprietary software firms have asserted that OSS (specifically that covered by the GNU Public License or "GPL") is a cancer that attaches itself to intellectual property" [Scacchi, 2007]. This critique comes from the fact that any modified version or software using GPL'ed software must use the GPL license, thus making it less attractive for commercial use and impossible to use in proprietary software. GPLv3 has taken further steps to assure the original intend of the GNU GPL license and it addresses recent challenges in computer software. There has been a lot of debate around GPLv3. It is worth noting that Linus Torvalds has not adopted GPLv3 for Linux, but is keeping GPLv2.

Most OSS licenses provide the means to protect the rights of software developer, and provide the means to protect the end-user as well [Cuéllar, 2005]. In figure 3.2 an overview of some of the common OSS licenses is presented.

	Notice of modification	Redistribution of the modified work	Original source code attached to the modification	Linking to closed source code	Liability notice
BSD	Yes	Yes	No	Yes	Yes
GPL	Yes	Only under GLP or LGPL	Yes	No	Yes
LGPL	Yes	Only under GLP or LGPL	Yes	Yes	Yes
MIT	Yes	Yes	Yes	Yes	Yes
MLP	Yes	Only under MLP	Yes	Yes	Yes

Figure 3.2: OSS License Features [Beard and Kim, 2007].

[6] http://opensource.org/about - Open Source Initiative

3.4 Recent Development

[Fitzgerald, 2006] labeled the recent development in the Open Source domain "OSS 2.0". It is argued that the OSS 2.0 phenomenon is significantly different from its FOSS antecedent. For example the planning phase in FOSS is thought of as "an itch worth scratching" [Raymond, 2001b] but in OSS 2.0 this is superseded by corporate firms considering how best to gain competitive advantage from OSS [Fitzgerald, 2006]. Greater commercial interest results in an increased amount of paid developers and more OSS produced for vertical domains. Companies can earn money by selling support or dual license their software. Hardware sales can also increase since the OSS which runs on the hardware has a lower cost than the alternatives of proprietary software.

In a study on Off-The-Shelf Component Based Development [Li et al., 2005] found that "Source code of OSS components was read frequently with very few modifications. OSS components were de facto used as COTS components, even if the source code was available." This suggest that the usage of OSS in component based development is not different from the usage of propriety components.

Intellectual property (IP) concerns around OSS is much debated. News stories on slashdot.org concerning legal issues are frequent. A list of some legal issues from 2007 can be found at the "Law & Life: Silicon Valley" blogg[7].

3.5 About the Apache Foundation

The Apache Software Foundation (ASF) was formed by the Apache Group in 1999. This group had previously made the Apache HTTP Server. The goal of the organization is to "provide organizational, legal, and financial support for a broad range of Open Source software projects. The Foundation provides an established framework for intellectual property and financial contributions that simultaneously limits contributors potential legal exposure"[8]. The foundation stresses that all who participate and/or donate code signs a contributor license agreement. This helps the foundations protect contributors legally. If a new project is suggested or donated, it enters an incubation state. The Apache Derby project would not have passed the incubation state if not SUN Microsystems had contributed developers to the project. IBM would have had all the developers working on the project, and having only one company running the project is against the Apache Foundations rules. This is to make sure not one company is in charge of everything in a project. It is also stated that when working in an Apache community you should take off your "company hat".

[7] http://lawandlifesiliconvalley.blogspot.com/2007/12/2007-top-ten-free-and-open-source-legal.html

[8] http://www.apache.org/foundation/

Chapter 4

Joining

Joining is normally viewed as part of Open Source. We have chosen to have a separate chapter discussing Joining literature, because it is the focus of the entire report.

Here we will define what we mean by Joining and present literature and previous studies related to Joining process. This is our State-of-the-art chapter.

4.1 Definitions

Before we can describe exactly what we mean by Joining, we have to define a few of the terms we will use.

- **The Cloud**
  This is what is best describes as the Cloud of people not actively participating in or contributing to Apache Derby in any way. They can be passive users of the product, and might even ask questions about the use of the product on the IRC channel or the derby-user@db.apache.org mailing list, but do not open JIRA issues or participate in technical discussions.

- **Joiner**
  This is the people who have initiated first contact with the development effort. This can be looking at the JIRA issue tracker, reading the developer mailing list, or even send a few questions on the developer mailing list.

- **Lurker**
  Lurker is usually a subcategory of Joiner. Before sending the first mail or answer a question it is common to have a period were you are on the mailing lists, lurking, while getting familiar with the project and comfortable with the technology. This is one step of what [von Krogh et al., 2003] describes as a Joining-script.

- **Newcomer**
  When a Joiner have had first contact with the mailing list, and start getting familiar with the community he will be defined as a Newcomer. He can then start participating in easier technical discussions, answer user questions and even contribute simple code or documentation.

- **Contributor**
  When a Newcomer have been around for a while, and gained some "geek fame" [Scacchi, 2007] he will become a Contributor. This is were a person starts becoming a valuable person in a project, contributing to the project instead of being the one in need of support. This does however not mean that you have to contribute code. Documentation and technical discussion will suffice to join this group.

- **Developer**
  A Developer is a person that have passed though all the other steps, and are now actively contributing code. We define both Contributor and Developer because we will use the term Contributor in our definition of Joining, and it is therefor important that the reader knows that they are not the same thing. A Developer is a code contributor, while a Contributor is contributing to the project in any other fashion.

- **Joining**
  This project will focus on Joining. We define Joining as the process were a person goes from being a Joiner to when they are Contributors. We choose not to include the full process of becoming a Developer, because this is a very time consuming task, and the time horizon on this project wont let us cover this well enough to get any satisfactory results. [Herraiz et al., 2006]

4.2 Personal Attributes to Consider when Joining Open Source

In this section we look at what is needed from the person that wants to join a project. We will look at the traditional Joining-script, as described by [von Krogh et al., 2003], and what motivates a person to contribute to an OSS project, as well as looking into what a Contribution Barrier is.

4.2.1 Joining-script

It has been observed that Newcomers to a project will have to demonstrate some level of skill and understanding when Joining a project ([von Krogh et al., 2003], [Lovgren and Racer, 2000]). [von Krogh et al., 2003] introduced the idea of a

Joining-script, but this was influenced by [Tilly, 1999]. Krogh claims that following certain rules when Joining a project will give you a much bigger chance of becoming an accepted member of the community. His proposition 1 says: "Participants behaving according to å Joining-script (level and type of activity) are more likely to be granted access to the developer community than those participants that do not follow the project's Joining-script.". He does describe a Joining-script for Freenet, were he outlines how a Joiner should show his skill, contribute to technical discussions and post concrete examples on the mailing list to be accepted. Inn [Holum and Løvland, 2007] you can see how we used the ForNewDevelopers page as a similar Joining-script, and how we got good feedback when doing so. [Bird et al., 2007] performed a statistical analysis of mailing lists of Apache web server, Postgres, and Python. He came up with the following hypotheses:

- Hypothesis 1
  Likelihood of attaining developer status will rise with tenure, peak at some point, and then decline.
- Hypothesis 2
  Demonstration of skill level, such as patch submissions and/or acceptances, will increase the likelihood of becoming a developer.
- Hypothesis 3
  Social status will positively influence attainment of developer status.

This supports the fact that following a certain behavioral pattern when Joining a project will help a new developer get accepted into the community. If the new developer can show a skill level and be social, the chance of being accepted will increase.

4.2.2 Motivation

There are several reasons for people to join OSS projects. [Scacchi, 2007], [Hertel et al., 2003] and [Wang et al., 2005] lists a few of these. Here we present a combined list of their results. We will later use these to see what Apache Derby can do to stimulate Joiners to keep them motivated to work on the project.

We have chosen to use the same motivational items under more than one heading. This is because these headings and motivational factors do overlap quite a bit. If you want a reputation in the Open Source community, we have to assume that you also want to be part of the Open Source community. This would again lead us to presume that you do believe in the inherent value of Free Software. The most important here is to see that there is a lot of different motivational factors

Motivational factors	
Self-Determination	Participate in a new form of cooperation. Participate in the Open Source scene. Improve my job opportunities. Get help in realizing a good idea for a software product.
Peer Recognition	Participate in a new form of cooperation. Learn and develop new skills. Share knowledge and skills. Participate in the Open Source scene. Improve Open Source products of other developers. Get a reputation in Open Source community.
Project Affiliation and Identification	Participate in a new form of cooperation. Participate in the Open Source scene. Get a reputation in the Open Source community. Distribute not marketable software products. Solve problems that could not be solved by proprietary software. Think that software should not be a proprietary good.
Self Promotion	Learn and develop new skills. Share knowledge and skills. Participate in the Open Source scene. Improve Open Source products of other developers. Get a reputation in Open Source community.
Belief in the Inherent Value of Free Software	Participate in the Open Source scene. Get a reputation in Open Source community. Limit the power of the large software companies. Solve a problem that could not be solved by proprietary software. Think that software should not be a proprietary good.

Table 4.1: Motivational Factors.

that comes into account when Joining a project. The personal motivation for staying can vary a great deal. We also find it important to point out that there is a difference between people payed to work with Open Source and hobbyists [Hars and Ou, 2002]. We consider this as important, since we in our previous research found that everyone with commit access in the Apache Derby project are employed to work on the project. The most obvious differences we see is in Hars and Ou's paper is Self-determination (92.6% hobbyists and 62.5% for paid OSS developers) and Selling products (3.7% for hobbyists and 83.8% for paid OSS developers).

4.2.3 Contribution Barrier

[von Krogh et al., 2003] also writes about the Contribution Barrier. This is a barrier that new Newcomers and Joiners have to overcome in order to contribute to the project. This barrier can be lowered by both the project and the person wanting to join. In table 4.2 you can see the list of Contribution Barriers identified by Krogh et. al when they studied Freenet. A common denominator that is easily identified is the modularity of the project. A project should provide a codebase or architecture that is clear and with high modularity. A Joiner should be able to find a module that suites his preferred programming language. It should be easy to integrate new modules and the interfaces between the modules should be clear.

KCB	Description
KCB1	ease of modifying and coding module
KCB2	the extent to which the potential developer can choose the computer language used to code for the module can vary
KCB3	ease with which to "plug" the module into the architecture
KCB4	the extent to which a module is intertwined or independently working from the main code

Table 4.2: Contribution Barriers [von Krogh et al., 2003].

A project can help lower their Contribution Barrier by improving any and all of the points mentioned in table 4.2. By doing this they will help ease the entry into the project for Joiners and Newcomers.

Joiners can themselves help lower the Contribution Barrier when Joining a project. The obstacles mentioned above will be hard to influence, but the background and specialisation of the Joiner may help lower the Contribution Barrier.

A Joiner with little previous OSSD experience and programming expertise, you can lower the Contribution Barrier by admitting this. Start with simple tasks, like reviewing documentation or running tests. A Joiner with a specialisation within a field of the project would normally be able to quickly contribute, because his expertise will help him lower his Contribution Barrier. After the first contributions, either on a simple task or in an area of expertise, a Joiner will gain knowledge about the project. This will help the Joiner contribute in other and maybe more advanced parts of the project later. Contributions will get the Joiner "geek fame", making it easier to later get contributions accepted into the project.

[Hertel et al., 2003] found that in the Linux kernel project the lack of time to be one of the biggest obstacles for participating. [Herraiz et al., 2006] found that volunteers Joining the GNOME project used 30 months to get enough knowledge about the project to be able to write code. For a Developer hired by a company to work on the project this time was significantly lower, but the exact time was not mentioned in the paper. This suggests that it is easier for a Joiner to be successful in Joining a project if he is hired by a company, and that this is because a volunteer, or hobbyist, does not have the same amount of time to spend getting familiarized with the projects. This also has to be considered as a personal Contribution Barrier that probably is very hard to avoid for a volunteer Joining the project.

4.3 Software Artifacts in OSSD

"Software informalisms are the information resources and artifacts that participants use to describe, proscribe, or prescribe what's happening in a FOSSD project."[Scacchi, 2007]. In table 4.3 the software informalisms, or artifacts, are listed. They are used in OSSD to coordinate the work, transfer knowledge and more. Some of these artifacts can be used to help Joiners and Newcomers. We will use this to see if Apache Derby can improve their artifacts to help Joiners get familiarized with the project, and reduce the time it takes for volunteers to contribute.

Different artifacts used in OSSD

Email lists	News postings	Discussion forum
IRC	FAQs	How-to guides
Scenarios of usage as linked Web pages	Traditional system documentation	Project property licenses
Wiki	Open software architecture diagrams	Intra-application functionality realized via scripting languages like Perl and PhP
Incorporate plug-in externally developer software modules	Integrate software components, modules, or scripts from other OSSD efforts.	Project related Web sites or portals
Software bug reports	Issue tracking data base like Bugzilla	Project specific Web sites
OSSD multi-project Web sites (e.g., SourceForge.net, Savanah.org, Freshmeat.org, Tigris.org, Apache.org, Mozilla.org)	Embedded project source code Webs (directories)	Project repositories (CVS / SVN)

Table 4.3: Artifacts from [Scacchi, 2007].

Part II

Participation

Chapter 5

Analysis of the Situation

In this chapter we formulate and describe our research questions and explain how they relate to both the research community and Apache Derby. We continue by mapping theory from the literature studied in chapter 4 to the practices in Apache Derby. We look into Joining-script, motivational factors, artifacts and possible Contribution Barriers.

5.1 Research Questions

At the very start of this semester we had meetings with one of the commercial companies working on Apache Derby were presented our previous report. They presented suggestions on what they wanted us to look at this semester. We wrote a RCA that can be seen in appendix C. This did not get signed, and as a consequence we got no more feedback from our client throughout the project. However, they helped us define a goal and gave us an area to focus on, which both us and them though of as interesting. We knew that at least one part of the Open Source project wanted us to research this, and even though we didn't manage to get a RCA signed, at least we knew they would agree on our research focus. One of the e-mails we got from the commercial company, and it's translation, can be found below.

```
************************************************************

Ola Nordmann - Kommersielt selskap wrote:

Jeg inviterer gjerne til et møte. Vi har snakket om at vi gjerne
ønsker at eksterne krefter undersøker litt om hvor lett Derby /
Java DB er å ta i bruk og hvor lett det er å bidra med kode,
bygge/teste systemet etc.  Er det ting som kan skriver eller
gjøres for å senke terskelen og dermed utvide community? Kanskje
```

```
det kan formuleres noen oppgaver i dette landskapet?

-Ola

This translates into:
Jhon Doe - Commercial company

I'll be glad to invite to a meeting. We have talked about how we
would like external people to research how easy DERBY / Java DB
is to start to use and how easy it is to contribute code,
build/test the system etc. Are there anything that can be written
or done to lower the threshold and thereby expand the community?
Can a thesis be formulated with this in mind?

-Jhon
************************************************************
```

We formed our research questions (RQ) based on this. The questions are meant to contribute both to Apache Derby and the research community. The research questions can be found below.

- **RQ1: Which obstacles are encountered by Newcomers to Apache Derby when Joining?** When Joining an OSS project there are different obstacles that can be encountered. We seek to find these. An obstacle can be be, but are not limited to, the lack of motivational feedback, bad or lacking tools or documentation. One thing to consider here is that different people will see different obstacles, and this makes it harder to identify generic obstacles that everyone can agree on. We will try to identify the obstacles that most Newcomers meet.

  By answering this question we contribute both to the research community and Apache Derby. If we can identify proved document and obstacles these, Apache Derby can use this information in a constructive manner. The same information can also be used by other researchers working in similar research domains to compare results and help generalize these.

- **RQ2: What can be done to ease the Joining process?** After finding obstacles when working on RQ1, we will look at what can be done to lessen the impact of these obstacles. If we can find an obstacle that might be completely removed, a plan for this will be compiled. Because of the duration of this project, and the time it will take to see any impact of changes done, measuring the outcome will not be part of answering this

research question

This research question is interesting to both the research community and the Apache Derby project. If we can get the changes implemented, and they work as intended, Apache Derby will benefit directly from this. If the changes are successful, the research community will have gained new knowledge on how to ease the Joining process in OSS projects. If the changes suggested is not a success, this will also help increase the knowledge base concerning the issue. The biggest problem here is if none of our changes are implemented. A result of this will be that this research benefits no one, except the researchers knowledge.

In our previous study we encountered some obstacles. When writing these research questions we defined ourselves as in the midst of the Joining process, and we wanted to use our own experience as well as the experience of others to help us in our work. Involvement in the community is a good way to answer our research questions and is also a basic part of CAR.

5.2 Data Collection

To gather the data needed to answer the research questions we used different metrics. We got involved in the community and gathered data through both our own experience and their communication channels. We used IRC and the mailing lists to look for possible obstacles that arose. Likewise we saw if solutions to these obstacles were posted. If solutions were not posted, we would try to find them ourselves.

Throughout the project a diary was kept so that the research wouls not be biased by retrospective reporting. We gathered statistics from the IRC metric by logging the activity on the #Derby channel. We payed close attention to the JIRA issue tracker. Most of the technical discussion happens there. This is also were the issues are created, and were the priority is assigned. To overview these processes and discussions has been of great value to us. The last metric we used is our autumn report [Holum and Løvland, 2007]. We found obstacles when working on it, and these has been transfered to this report.

5.3 Joining-script

In section 4.2.1 we presented the theory of a Joining-script, and how a participant can follow this to easier get accepted into a community. An important factor for a project wanting to attract new Developers is be to publicly post their Joining-script, and make it easily accessible for Joiners wanting to become

Newcomers. When we started to work on Apache Derby last fall we did find good documentation describing Joining in Apache Derby. It was easy to find information about the e-mail lists, IRC and the ForNewDevelopers document. The ForNewDevelopers document explain what they want Joiners to start with before they begin working on JIRA issues. We did follow this document ourselves, and got good and fast feedback on the mailing list as you can see in the JIRA issue tracker[1]. We do describe some hypotheses about Joining-script in section 4.2.1. When reading the mailing lists of Apache Derby one can see that these hypothesises hold for the Apache Derby project. We have included a request to vote Jørgen Løland a committer. Here one can see that it is both his social and professional skills that they comment on.

From the data presented above, and our own experience, we conclude that the availability of a Joining-script in Apache Derby is not an obstacle for Joiners who want to contribute to the project.

```
************************************************************

Subject: [VOTE] Jørgen Løland as a committer
From: Rick Hillegas <Richard.Hillegas@Sun.COM>
Date: Thu, 03 Apr 2008 10:03:18 -0700
To: derby-dev@db.apache.org

Please vote on whether we should make Jørgen Løland a Derby
committer. The polls close at 5:00 pm San Francisco time on
Thursday April 10.

Jørgen has contributed significantly to adding replication
functionality to Derby. Although he readily seeks and
cheerfully incorporates community feedback, his work does
not need supervision by other committers. In addition,
Jørgen fields questions on the user lists and his
interactions with the broader community are respectful,
thoughtful, and thorough.

Regards,
-Rick

************************************************************
```

[1] https://issues.apache.org/jira/browse/DERBY-3165

The phrase Joining-script is not used by the Apache Derby community, but they got it in the form of linked web pages. The Wiki page ForNewDevelopers[2] is approximately the same as the Joining-script described by [von Krogh et al., 2003].

5.4 Motivation

According to our previous research on Apache Derby, all committers are employed by commercial companies. [Hars and Ou, 2002] writes, as mentioned in section 4.2.2, that 83.3 % of payed Developers are motivated by selling products. However, selling products are not necessarily the motivational factor for Joiners, and it differs with each individual. We will not try to measure what motivates a single Developer in Apache Derby, but rather see if Apache Derby stimulates the motivational factors described in table 4.1. Below we look into those five factors.

Self-Determination
The Apache Derby project do fulfil the criteria for this factor. If you join as a free agent you will be able choose what part of the project you want to work on. We cannot say what the situation is for the payed Developers, but from what we see on the mailing lists we got the impression that they decide what parts of the project to focus on. Apache Derby is a mature and well know project and has corporately supported versions, known as Cloudscape and Java DB. Working on this well known project will increase the Developers job opportunities. The Apache Derby community is positive and open for new ideas. An example of this can be seen below, were we show a part of an e-mail correspondence when a new idea was suggested to the community.

```
************************************************************
Hi Sun,

This is great to hear! Peter Yuill is also working on adding
spatial datatypes to Derby. It may be possible to coordinate
your efforts and divide the feature between the two of you.
Here's a pointer to a recent email thread in which Peter
describes his work:
http://www.nabble.com/Spatial-Functionality-td17042147.html#a17042147
I encourage you to contact Peter. I believe that the community
will be very eager to support the two of you in this effort.

Thanks!
-Rick
```

[2] http://wiki.apache.org/db-derby/ForNewDevelopers

```
Sun Ning wrote:
> Hello!
>
> I am willing to develop a spatial extension for Apache Derby
> database. This will enable derby to store and manage
> geospatial data in the way of OpenGIS standard: Simple
> Feature Access. That means some geometry types and build-in
> functions should be added to the engine. After that, we can
> use Derby as a geospatial data source, at the backend of a
> portable mapping software or a web mapping site.
>
> I will do this in the later three months. So I just post this
> to the mailing-list to get some suggestions from you.

************************************************************
```

Peer Recognition

It is a very active group of people on the Apache Derby mailing lists. On average there are 60 mails a day on the developer, user and commit lists combined [3]. The response time is short, and the community big, with 207 subscribers to the developer list and 440 subscribers on the user list. This means that there is a lot of people overlooking the progress of the project, that can notice good work. So far the feedback people have gotten on the mailing lists have been positive and constructive. We conclude that the motivational factor of Peer Recognition is fulfilled.

Project Affiliation and Identification

Sun Microsystems and IBM are selling Apache Derby as Java DB and Cloudscape, respectively. These are well known products and companies in the IT business. The Apache foundation is also well known and ensures that the project is lead following certain rules, which increase the integrity of the project. The motivational factor of Project affiliation and Identification is not an obstacle.

Self Promotion

The Apache Derby project is a mature project. It has stable releases, a big core of people working on it and the necessary tools are in place. This means that the most trivial problems are all solved, or will probably be solved by one of the experienced Developers as soon as they find them. This can make it harder for Joiners to start contributing to the project, and might make the motivational factor Self Promotion hard to achieve.

[3] http://people.apache.org/~coar/mlists.html

On the other hand, the same things that makes it hard to get started makes it a very good place to be when you have made it past the first thresholds. When a person have achieved the knowledge needed to actively participate and work with the community he/she will have the benefit of working with experienced programmers from two of the biggest computer companies in the world. And that can be used in Self Promotion.

Belief in the Inherent Value of Free Software
[Hars and Ou, 2002] says that 83.8% of paied programmers have selling products as a motivational factor. Seeing that the committers in Apache Derby are payed to work on the project, this motivational factor is not very important. However, we do not feel that there is any reason why this fact would keep others that want to join, because they believe in the inherent value of Free Software, away from the project. The Derby project is part of the Apache foundation, and they follow strict rules to ensure that the project is indeed an Open Source project, following the norms of OSSD and the Apache Open Source licence. By being an Apache project Apache Derby can attract people who have the Belief in the inherent value of free software.

5.5 Contribution Barrier

When we looked further into this issue, we used table 4.2 as our starting point. [von Krogh et al., 2003] have developed this list of Contribution Barriers. This was when they studied Freenet[4], an OSS project developed with the GNU GPL licence. We have found that some of the Contribution Barriers that they found here can not be applied in the same way to Apache Derby, because the programs developed are very different from each other, both in usage and architecture. We did find that Apache Derby is module based, and that working on one module can be done without it interfering too much with other modules. From this we can conclude that KCB1 not an obstacle.

KCB2 states that a potential developer should be able to choose what programming language to work with. Apache Derby is a database coded in Java, and changing the language would not be an option. If the Joiner or Newcomer does not know Java he will have to learn it before he can contribute code to the project.

Being a database, it is not easy to plug modules into the architecture (KCB3). In a top layer application project modules can be everything from new functionality through new buttons and windows to new GUI elements. In the Apache Derby project they do follow the SQL standard strictly, and changes to the SQL syntax must comply with the standard. Even though new functionality is added

[4] `http://freenetproject.org/`

with new releases this report will not look further into what obstacles can be found when integrating new modules in Apache Derby. Introducing new functionality to a mature project is not an endeavour a Newcomer should set forth.

We do not have enough knowledge about the architecture of Apache Derby to know how close modules are intertwined with the main code (KCB4). Let us assume a Developer want to change or add an encryption algorithm for encrypting the database. For what we know this can be done without too much interference with the rest of the code, but we do not know this for certain.

"The problems facing distributed software development teams are reflected in Conway's Law, which states that the structure of a product mirrors the structure of the organization that creates it."[Crowston et al., 2007]. Apache Derby once was a closed source project called Cloudscape, developed by IBM. This was done by a geographically close software development team and is reflected in the code. This is a Contribution Barrier. A possible challenge here will be that some of the knowledge about the code of Apache Derby are not made explicit, but exist tacit in the heads of the former Developers. We believe the knowledge is still in the community, since many of the Developers are from the original development team of Cloudscape.

The Contribution Barriers found by [von Krogh et al., 2003] can not be applied directly to Apache Derby. The main focus of the Contribution Barriers found by Krogh has to do with the modularity of the code. Any reason that would keep a person from contributing to the project will be classified as a Contribution Barrier. The difference between a Contribution Barrier and an obstacle will be the abstraction level. A spelling error in a document can constitute an obstacle, but wrong or bad documentation will be the Contribution Barrier. This is in line with the idea of a Contribution Barrier presented by Krogh, even though it is changed to suite Apache Derby and our research method.

5.6 Artifacts

As a mature project, following the rules of the Apache Foundation, the artifacts discussed in table 4.3 are naturally in existence. We will therefore discuss the artifacts inadequately used, rather than trying to find artifacts not used at all.

When starting to work on an Open Source project, the first thing one would do is to sign up to the e-mail lists and join other communication channels they might have. This can be in the form of forums or IRC. We found their e-mail communication to be very good and useful from the start. We did not find a discussion forum for the project, but we quickly realised that they didn't need one, because it would all be discussed on the JIRA issue tracker and on the mailing lists. The communication would then be posted on the Internet, for everyone to read. This gives the same functionality as a web-based forum. We

did not find the same level of activity on their IRC channel. As Newcomers to the project we thought IRC would be a good place to start asking simple questions and getting basic help when building the project. Getting more people to actively use IRC would help Newcomers.

After a Joiner becomes aware of all the communication channels, the Joiner would normally start looking for ways to contribute. Here we looked for guides, Wiki and the likes. We did find good guides and user manuals for the different versions of Apache Derby, but we found much less information for Developers. It was hard to find information on architecture, structure and code conventions. We did find some discussions on code convention on the JIRA issue tracker, but these issues were not closed. Gathering more info on what documentation a new Developer needs and usually looks for would be useful. The ForNewDevelopers page mentioned earlier gives help to get started, but it does not give much information on how to do the tasks suggested.

Chapter 6

Iterations

In this chapter we start by presenting the obstacles found in our previous work. We then continue to present our iterations. The research model used in all our iterations is the one described in [Davison et al., 2004]. This is reflected in the structure of this chapter, were we have the five phases; Diagnose, Action Planning, Intervention, Evaluation and Reflection as part of each iteration.

6.1 Our Previous Work

In our previous study on Apache Derby [Holum and Løvland, 2007] we started our Joining process. In that study we got familiar with OSSD processes in Apache Derby and encountered an outdated document describing the build process of Apache Derby. We got our suggested updates to BUILDING.txt accepted into Apache Derby repository[1]. Below is a list of Known Obstacles we encountered during our involvement last year.

- **KO1: Installing correct programs, correct versions and setting up the environment variables.**
  This involves configuring, downloading and setting up Ant, the different and correct Java versions and adding the correct $CLASSPATH as described in BUILDING.txt and Readme.htm. Ant is a building tool for Java. #CLASSPATH is an environmental variable. It can be set locally in a shell or command line. To set the variable globally the MS Windows environmental variable window or the *NIX startup file is used. The function of the #CLASSPATH is to tell the Java Virtual Machine where to look for user-defined classes and packages when running Java programs.

- **KO2: Hard for Newcomers to answer user questions**
  The questions either required an in-depth understanding of Apache Derby

[1] https://issues.apache.org/jira/browse/DERBY-3165

or were answered too quick. As a result of this we never got the chance to answer user questions. By the time we had found an answer somebody else had already given one.

- **KO3: Lack of activity on IRC**
  We noticed almost no activity on IRC when we first joined the project last semester. Users asked questions, but answers were few. When we asked questions about our test logs, we got no answer. We think the low IRC activity leads to even less activity. If the activity was high, checking it frequently to see if anything new had happened, would be part of a Developers routine. When the activity is low, there is no tangible reason to check the channel. This leads to the long response time on IRC.

- **KO4: Availability of software**
  When building an old version of Apache Derby (10.2) we had problems finding the required java versions, since Sun Microsystems no longer supports Java 1.3.

KO2 is a result of the lack of hobbyist working on Apache Derby. In [Holum and Løvland, 2007] we found that most of the Developers worked for either SUN or IBM. We might find more questions to answer this semester since there has been an increase in applications for Google Summer of Code projects. Counting 12 in total as this is written.

The current version is 10.4. We investigated an issue regarding Apache Derby 10.2 when we discovered this obstacle. Because the development of 10.2 is discontinued, we decided not to look further into **KO4**.

6.2 First Iteration

In our previous study we did one cycle of the CPM. This iteration is a continuation of our previous research done in [Holum and Løvland, 2007].

6.2.1 Diagnosis

As seen in the above list (**KO3**) we found that activity on IRC is a probable obstacle. We noticed an increased use of IRC this semester. Many of the new names are students that have signed up for Google Summer of Code(GSoC). From what we saw Newcomers came to IRC to get basic help and to have an informal first contact with the project. There were a few of the experienced Developers there as well, but the reaction time on questions varied from a few seconds to many days. Getting more reliable activity on IRC would make it more interesting for Developers and Newcomers to stay in the #Derby channel. This would again lead to more people wanting to stay in the channel, hopefully

leading to even more activity. There could be a lower threshold for asking a questions about building and setting up the environment on IRC than on the mailing list.

6.2.2 Action Planning

We made an action plan for this iteration. The plan can be found in table 6.1.

We plan to send an e-mail to the Derby-dev list letting everyone know that we will be available on IRC to help Newcomers getting started. With all the new people signing up for GSoC, we hope top be able to help them with problems they might have building the code, running the tests, navigating the documentation and setting up the environment. This will give us a dialog with the Newcomers and maybe we will identify some obstacles which Newcomers encounter when starting working on Apache Derby.

Table 6.1: Plan for the First Iteration.

6.2.3 Intervention

To start this intervention we sent the e-mail posted below to the developer mailing list, where we inform everybody that we would be available to help Newcomers with their questions.

```
**************************************************************

Hi,

me and Henrik would like to help the new people if they have
any difficulties when starting working on Derby. We'll mainly
be available on IRC 8:00-14:00 GMT and try to answer any
questions we're able to. Perhaps we can find something that
will make starting working on Derby easier.

We don't have an in depth-understanding of the code, but have
been lurking for a while and believe we know where most
information can be found. We've build and run tests on both
linux and windows with a variety of jdk's.

Regards,
Svein Erik (sveinelo)
```

```
Henrik (Hebla)

PS.
To send messages directly to anyone on IRC you have to register
your nick on freenode.

*************************************************************
```

After lurking on IRC for a week, we saw no questions directly to us or concerning building, testing, finding documentation or setting up the environment. But we noticed an increased use of IRC. We got a quick positive answer on the mailing list by a committer named narayanan, and he also joined IRC were he had a short discussion with kmarsden. There were some Newcomer questions in the weekend, but unfortunately we were not online to help. The Newcomers asking questions in the weekend got help from other participants, so there were not a lack of response and they got the answers they needed.

6.2.4 Evaluation

We saw that a lot of the Joiners, specially the ones that were waiting for answers on their GSoC applications, were asking general question on IRC. kmarsden was answering these questions rapidly. She was the person responsible for handling the GSoC applications, and assigning mentors to the people that got accepted. Shortly after we sent the e-mail to the developer list Narayanan answered the e-mail before joining the #Derby IRC channel. This led him to have a short talk with johnemb to coordinate testing of the 10.4.1.1 release candidate. Here they encouraged everyone on the channel to have a look and help with the testing of the release candidate, something we considered for our next diagnose phase. Narayanan was a regular on IRC for a long time after this, even though he was not registered in our logs earlier.[2]

We did not have adequate data to confirm that our effort to increase the IRC activity was successful or if it has eased Joining. We therefore decide to continue our activity on IRC and monitor the channel throughout our project.

One of the reasons why we found it hard to evaluate if our actions had an impact is the increased activity of kmarsden. Her response time on IRC has been very low since the applicants for GSoC joined the channel. Her knowledge of the codebase and good overview of the issue tracker left us a bit redundant.

[2]kmarsden, Narayanan and johnemb are IRC nicknames used on the #Derby IRC channel

6.2.5 Reflection

When we diagnosed IRC to be an obstacle there were few Newcomers to the project. Almost all activity would be from active members of the community that seems to have a good understanding of the program, issues and codebase. When looking back we realise that IRC communication among Developers experienced in the Apache Derby development process might not need the communication channel provided by IRC. The little activity we saw on IRC would mostly be users joining the channel to ask questions about use, not development. When the Joiners from the GSoC project started asking questions on IRC we could see a much better response time, specially from kmarsden. This leads us to believe that they are not neglecting the opportunity to recruit and motivate possible Newcomers on IRC. They do step up the IRC activity when needed, instead of having a constant focus on it, even when the amount of Joiners and Newcomers are low.

From this we will deduct that an increased IRC activity might help catch the interest of few Developers on a normal day, but the decreased response time when there are visible Joiners and Newcomers in the project makes IRC much less of an obstacle than first anticipated. We did not have enough data to answer either RQ1 nor RQ2, and therefore we decided to continue our involvement with the Apache Derby community. We also decided not to change the research questions.

6.3 Second Iteration

In our first iteration we got little interaction with the community. In this iteration we wanted to make our presence more visible to the community. We chose to work on issues from the JIRA issue tracker this intervention.

6.3.1 Diagnosis

When we started our second iteration the community was working on the 10.4 release candidate. Testing this was a natural starting point for us as Newcomers. In our previous work we found an obstacle while setting up the environment and building our own binary files for Apache Derby(**KO1** in section 6.1). We found that it was more time consuming than expected to set up the test environment and run the test suites. We did not run the JUnit tests last semester, and seeing how these are a high priority to the project, mapping obstacles concerned with both JUnit and the old harness suite are of value to the research questions.

Tasks	Description
T1	Choose an issue from JIRA issue tracker flagged as Newcomer.
T2	Actively seek out Newcomers and ask if they need any help.
T3	Run tests on the 10.4 release candidate, analyze results and report findings back to community
T4	Improve on the architecture description and compose a diagram of it.
T5	Improve upon /java/testing/README.htm , which is a document describing how to run the test old harness test suite with link to how to run the JUnit test suite.
T6	Check if the $CLASSPATH environment variable can be set automatically, either with a script or with the build tool ANT.
T7	Convert a test from the harness suite to JUnit.
T8	Choose a documentation issue from JIRA issue tracker and complete it.
T9	Run tests on Ubuntu 7.x with IBM JAVA 6 on 10.4, analyze result and report back findings.
T10	Improve upon DERBY-3601.
T11	Choose a build issue from JIRA issue tracker and complete it.
T12	Improve upon DERBY-1671.
T13	Test JMX support in the release candidate.

Table 6.2: Possible Tasks for the Second Iteration.

6.3.2 Action Planning

During the first iteration we compiled a list of suggestions for what we could do in our second iteration. The list of possible tasks is presented in table 6.2. Issues from the JIRA issue tracker have a number in this format: DERBY-XXXX. All references to such issue can be found in the JIRA issue tracker of Apache Derby[3].

From the list of possible tasks in table 6.2 we deduced a plan for our second iteration. We chose to work with T3, T5 and T9. The plan is presented in table 6.3.

In this iteration we will build and test the 10.4 release candidate (T3). While doing this we will look at the Readme.htm document(T5). This document describes how to setup and run the test suites. Looking at Readme.htm is a natural step when building the release candidate. This means that T3 and T5 should be possible to work on in parallel. Afterwards we will be running tests on the release candidate on Ubuntu 7.x with IBM JAVA 6, and report our findings(T9). This was requested on the mailing lists. We plan to do the following tasks: T3 - Run tests on the 10.4 release candidate, analyze results and report findings back to community. T5 - Improve upon /java/testing/README.htm , which is a document describing how to run the test old harness test suite with link to how to run the JUnit test suite. T9 - Run tests on Ubuntu 7.x with IBM JAVA 6 on 10.4, analyze result and report back findings.

Table 6.3: Plan for the Second Iteration.

6.3.3 Intervention

When we started this intervention we could see on the mailing lists that T9 was already resolved. The errors found by the reporter were not real failures, but showed up because of a fault in the reporters build environment. We therefore tried to find another issue concerned with testing 10.4. This time we used the JIRA issue tracker to find a Newcomer issue to fulfil T1. This was instead of picking the issue from the mailing lists. We found DERBY-1342 which was a Newcomer issue, concerned with testing. This became the main issue of this

[3]http://issues.apache.org/jira/browse/DERBY

DERBY-3638	java/testing/Readme.htm location of derbyTesting.jar
DERBY-3642	Update and rectify the Derby tutorial
DERBY-1342	Run derbyall suite with -Dframework=DerbyNetClient and analyze failures.

Table 6.4: DERBY issues Second Iteration.

intervention. T1 and T3 could be combined, and as you will later see, this was what we did. Table 6.4 contains the JIRA-issues involved in this iteration and table 6.5 contains a short version of our diary. This is included to show the important dates of this intervention.

To run the test suite you have to sett an environment variable called $CLASSPATH. This had to be done before we could work on the main tasks T3 and T1 (DERBY-1342). Setting the $CLASSPATH variable is listed as a possible obstacle from our earlier work. Because of this we decided to try to make a script which automatically sett it. The $CLASSPATH variable have to point to every .JAR file in the Apache Derby release you want to work with. You can either set it in your windows environmental variables, making it a global variable, or you can sett it in you command window to make it a local variable for the command windows lifetime. We encountered a few problems setting this variable automatically. You can have different builds of Apache Derby and you can have different bin releases, a sane build or an insane build, on the same computer. This meant that the script would have to take input parameters, and making a script see the differences between the various Apache Derby jars proved to be a big challenge. Seeing that the use of the script would be troublesome, and how hard it would be to make the functionality of the script good enough to be usable and intuitive, we concluded that it is easier and faster to create the variable manually. Simply making a text file with the different $CLASSPATH variables you might need and then copy-pasting from this file to the command line, is much easier. This seems to be the de-facto standard in the community, and we therefore chose not to post our observations on the mailing list. This work was connected to T6. We updated the example on how to set the $CLASSPATH in the testing documentation(DERBY-3638) while working on this task. To update the test documentation fulfills T5. This document did not need any other updates, so T5 turned out to be less work than expected.

When we started working on T3 there was no binary distribution released for 10.4. Therefore we compiled the branch ourselves and ran tests on the self built binaries. When we ran the test suites on our binaries we got so many failures that we believed we had done something wrong when building them. We started analyzing the test suite output for clues to what might be wrong. It looked as if connections to the default port for Apache Derby server were refused, but we

Date	Short Diary
19.04.08	Intervention started Started testing of 10.4 release candidate
20.04.08	Discovered inconsistency in location of derbyTesting.jar.
21.04.08	Created DERBY-3638, patch submitted.
23.04.08	DERBY-3638 - Patch accepted.
26.04.08	Release of Derby 10.4.1.3 - bin distribution.
28.04.08	DERBY 3642 - worked on issue. DERBY-1342 - Tested release candidate again with - DerbyNetClient set.
29.04.08	Started testing of the 10.4-bin distribution.
30.04.08	-Dframework=DerbyNetClient parameter made the tests hang.
01.05.08	Finished work on DERBY-1342. Considered reopening DERBY-3638 - derbyTesting.jar. Looked into working on DERBY-1671 - junit.jar.
02.05.08	Got verification that patch DERBY-3638 was correct.

Table 6.5: Short Diary.

had no firewalls that could block the connection requests, on our machines. This puzzled us for some time and during this analysis of failures we found the time to help with another issue (DERBY-3642).

DERBY-3642 deals with an Apache Derby tutorial which needed updates because of the upcoming release of Apache Derby 10.4. To get a better understanding of how Apache Derby works and perhaps understand why we got all those failures when working on T3, we decided to work through the tutorial and at the same time to improve it. This would fulfill T8. We first discussed some changes on the IRC channel and reported the result from the discussion in the JIRA issue tracker. Unfortunately this did not help us much in understanding the failures we got from testing our self build binaries. During this work the binary release was ready and made public. We decided not to dwell longer on the analysis of the failures on our self built binaries, and instead run the same tests on the now finished and released binaries. This reduced the possibility of it being our own built environment creating the failures.

Remember that our main task was to run the test suite with a special parameter according to DERBY-1342. Now that the community build binaries were available we downloaded them and ran the test suites without the special

parameter to be certain that they ran cleanly, which they did. We then went on and ran the test suites with the parameter -Dframework=DerbyNetClient[4]. This resulted in that the derbyall suite hung and never continued when it came to a test called store/streamingColumn.java. The derbyall suite was run on two different operation systems, Linux and Windows XP, with Java virtual machine 1.6 and both machines hung on the same test. After we manually stopped the test runs we started to analyze the outputs created by the suite, but we did not understand all of what we were looking at. We therefore asked for help on IRC. After some discussion with a Developer on IRC, we got the impression that spending time running and analysing the results of this test would not be very useful to the community. They are trying to move away from the old harness test suite, and continuing working on this issue would be a step backward rather than forward. Therefore we chose to stop working on this issue and withdraw to do our evaluation and reflection.

6.3.4 Evaluation

One miscalculation we did in this iteration was that what we though would be the main task did not turn out to be a real issue after all. The result of that was that we could not strictly follow our plan from the Action Planning phase, and the iteration sometimes felt unstructured. To avoid this we could have done a more thorough analysis of T9, but since the analysis of this problem was the issue, doing so would be the same as starting the intervention. Another thing to mention is the fact that the action planing phase took some time, since we had a lot of possible tasks to evaluate. T9 was a live issue on the mailing lists, so it was resolved in the time from when we found it to when we had finished our action plan. Instead of T9 our main task turned out to be a combination of T1 and T3, and exists in the JIRA issue tracker as DERBY-1342.

In our previous report we found that setting the $CLASSPATH variable to be an obstacle for Joiners and Newcomers (**KO1**, section 6.1). As mentioned in section 6.3.3 we encountered this obstacle again when we worked on the main task. Because of this we made an effort to remove the obstacle by working on T6, which was not a part of the original plan. Even though we did not succeed in finding a solution for T6 we gained more experience about challenges connected to **KO1** and T5. The improvement on Readme.htm (DERBY-3638, T5) would not have been possible without working on T6. T5 turned out to be a smaller task than expected. The Readme.htm file was not as outdated as we first thought, and the only update we chose to add a patch for was the location of derbyTesting.jar. The interactions with the community about the location of derbyTesting.jar was valuable to us. We got to interact with the community in a positive manner.

[4]The DerbyNetClient parameter runs the test using the server and the Apache Derby client driver (as opposed to using the embedded driver)

DERBY-3642 was another issue that enabled us to interact with the community. Here we helped a Newcomer on a documentation issue concerning the tutorial. We suggested changes, and our findings were included in the patch submitted. This patch got committed, but the issue is not yet closed. Even though the issue is not closed we are satisfied with the work we did on this issue. Since the patch got committed this issue satisfies T8.

When working on DERBY-1342, running tests on the release candidate, we got a huge error log. At first we thought we had done something wrong, because of the amount of failed tests. The other work we have done this iteration was connected to all the failed tests. Looking back we can say that trying to figure this out have given us information and experience working with Apache Derby. We did find all the failures to be an obstacle to us as Newcomers. The issue is flagged as a Newcomer issue, but it gives no information on how to handle the test logs or what to expect when running the tests with the special parameter. We had a hard time trying to understand what was actually happening, and in the end when we were asking for help on IRC, we got told that they were trying to move away from the old test suite, so our energy would be better spent elsewhere.

Tasks	Description
T1	Choose an issue from JIRA issue tracker flagged as Newcomer.
T3	Run tests on the 10.4 release candidate, analyze results and report findings back to community
T5	Improve upon /java/testing/README.htm , which is a document describing how to run the test old harness test suite with link to how to run the JUnit test suite.
T6	Check if the $CLASSPATH environment variable can be set automatically, either with a script or with the build tool ANT.
T8	Choose a documentation issue from JIRA issue tracker and complete it.
T9	Run tests on Ubuntu 7.x with IBM JAVA 6 on 10.4, analyze result and report back findings.

Table 6.6: Tasks Worked on in the Second Iteration.

We are satisfied with T1 and T3 as our main tasks, because they led us to the other tasks and issues we worked on this iteration. On a personal level we did learn a lot. However, we can't say that this iteration have been as usefully to the community as expected. We helped on some minor issues, and had some useful communication on the mailing lists, but our findings turns out to be on a personal level, and was not worth reporting to the community. We do have a

test log for DERBY-1342, but did not post this on the JIRA issue tracker. This because we felt that we did not manage to analyse it thoroughly enough, and we did not want to make unnecessary noise. We could have used more time trying to understand these test logs, but after the feedback we got on IRC, the use of this would be minimal. We decided not to work further with the old harness test suit and stop the intervention.

In table 6.6 all the tasks from this intervention are presented. It is a subset of the tasks in table 6.2, which was used in the action planning stage. We have marked the planned actions with bold font in the first column. As mentioned earlier we ended up working on tasks which were not a part of our original plan. There were two reasons for this. The first reason was that some of the suggested tasks in the action planning phase were overlapping or closely related to one another. The second reason was that our planned main task (T9) turned out not to be something we could work on, and we instead chose a combination of T1 and T3, realized in DERBY-1342, as our main tasks. The changes to the initial plan made our intervention unstructured at times.

6.3.5 Reflection

Four obstacles were found in this iteration. First we present the obstacles and then possible solutions to mitigate or remove the them.

We found the $CLASSPATH variable to be problematic. Setting this variable for a Joiner, and understanding how it works is not as trivial as one might expect at first. Also the fact that it change every time you have a new build of Apache Derby helps complicate the situation. Possible mitigations for this would be to describe even better how the variable works. Giving a deeper understanding will make a Joiner able to understand how this variable works with Apache Derby and edit it to the wanted use. Also giving the tip to save the various versions of the variable in a text file could save a Joiner both frustration and time.

Another obstacle encountered while working on this intervention was incorrect documentation. This made us spent some time trying to figure out what was wrong. We found that outdated or wrong documentation is an obstacle for a Newcomer in a project since this could result in either wasting time on unintelligible errors or that the Newcomer loose interest and leave the community. This makes it very important to keep the documentation updated. One way to improve this would be to have a person responsible for the documentation effort. It is important that this person will focus on the documentation for Apache Derby Developers, not only for the users.

The third obstacle encountered in this iteration was the test result log. When working on DERBY-1342 we found it challenging to analyse the test logs for several reasons. The directory structure and file structure was hard to understand. There are some information concerning this in the Readme.htm file, but even

after reading this, navigating the test result logs turned out to be a challenge best. When we found the files we though we were looking for we encountered yet another problem. The log listed 36 tests that failed in the test suite derbyall. Each of the tests that failed had multiple errors in their logs, and starting to analyze this amount of failures is not a trivial task for a Newcomer. The fact that the error logs requires an in-depth knowledge of the test suite and the code base makes this task unsuitable for a Newcomer. To remove this obstacle the community should encourage Newcomers not to start working on the old test suite. The focus of Newcomers that want to start working on tests should be on JUnit testing. Apache Derby are currently converting many of the old harness tests to JUnit, so this would help a Newcomer get to know what will be instead of what have been.

We encountered the obstacle with the test logs while working on an issue flagged as Newcomer. When working on this issue we did however find the level of knowledge needed to exceed what we would expect from a Newcomer issue. We found it very demoralizing not to have any success in progressing on an issue pictured to be solvable by a Newcomer. This made us feel that the contribution barrier for this project might be even higher than expected, and quite discouraging. To ensure that the correct flags are put on issues could help Newcomers a lot. We do not have any good suggestions on how to ease this obstacle, and it looks like the community uses their own judgement when it comes to flagging an issue as Newcomer.

In this iteration we worked on the old harness test suite. A Newcomer to Apache Derby might start to work on another area of the project. We decided not to withdraw from the community and continue our research to see if we could find other obstacles in a different area.

6.4 Third Iteration

In this iteration converted an old harness test to JUnit. We continued to use the same research model as in the two first iterations, so the outline of this chapter will be the same as seen above. The research questions have not been changed.

6.4.1 Diagnosis

During the first two iterations we learned that most of the Joiners to Apache Derby were doing so through the GSoC project and most of them had started working on converting tests from the old harness suite to JUnit. We wanted to see if we could find any obstacles when carrying out the same thing as the GSoC participants.

6.4.2 Action Planning

In the previous iteration we did not include the community in the design of our action plan. One consequence of this was that our main task turned out not to be a real task. Therefore we wanted to use the mailing list to get suggestions on what test to convert to JUnit. This would ensure that our work was relevant for and needed by the community. A part of the e-mail correspondence can be seen below.

```
************************************************************

Svein Erik Reknes Løvland wrote:
Hi,

I want to convert a test from the old harness to JUnit.
 Any suggestions on which test I might work on?

Narayanan wrote:
http://issues.apache.org/jira/browse/DERBY-2514
tagged as Newcomer, seems fairly easy for starters,
You might also want to look into closely at the comments by Myrna,

Narayanan wrote:
You can also take a look at
http://issues.apache.org/jira/browse/DERBY-1903
This is a bigger one than 2514, I guess if you want to look
 at a simple one first 2514 is better.

************************************************************
```

DERBY-2514 was selected. The full correspondence can be found in the e-mail archive[5]. Our plan for the third iteration can be found in table 6.7.

6.4.3 Intervention

First we started by looking at the code connected to the issue at hand. We found the test from the old harness suite easy to read and understand. The JUnit test covering the same area, but not all of it, was not as intuitive to read. This was because the JUnit tests made use of some parts of the Java API which we were not familiar with and we had not yet read the JUnit documentation on the Wiki. We let the community know that we were going to work on the issue and that

[5] http://www.nabble.com/I-want-to-convert-a-test.-to17338834.html#a17338834

In this iteration we will convert a test. From the discussion with the community we decided to look at DERBY-2514. DERBY-2514:convert lang/closed.java to JUnit From the discussion around the issue in the JIRA issue tracker we understand that most of the tests in lang/closed.java are covered by the JUnit tests in ClosedObjectTest.java, with two exceptions. - Exception 1 : The DERBY-62 test case Exception 2: ClosedObjectTest does not test reading of DatabaseMetaData obtained earlier, after a connection is closed. Our plan is to convert the two exceptions into JUnit.

Table 6.7: Plan for the Third Iteration.

we were in the process of learning about the JUnit framework . We were given a link to the Wiki with information about converting a test (see figure 6.1).

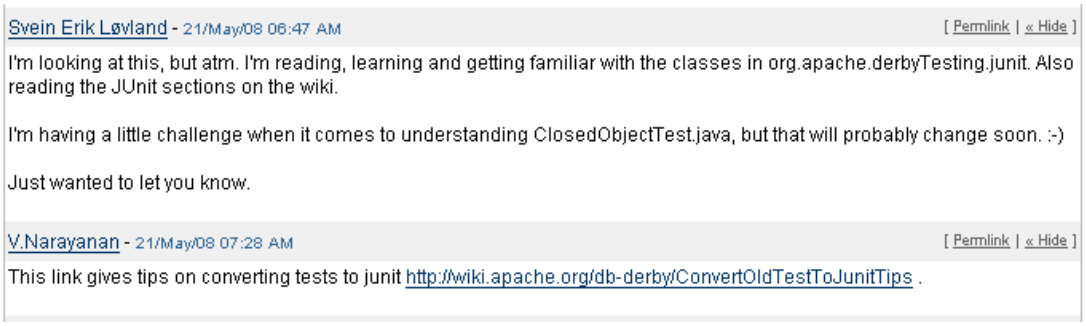

Figure 6.1: Screenshot from the JIRA issue tracker #1.

The documentation around JUnit was spread out on several Wiki pages, each with a specific focus (see appendix B). We completed a tutorial on JUnit and how it is used in Apache Derby. This was quite useful and prepared us to use the existing and extended JUnit framework of Apache Derby. In our opinion the framework was intuitive and it was quite easy to understand how to use.

After preparing ourselves for the actual task we made a patch to solve the issue and attached it to the JIRA issue for review (see figure 6.2). We did not set the patch available flag since it was not intended for inclusion in the code, but

Figure 6.2: Screenshot from the JIRA issue tracker #2.

rather a suggestion on how to solve it. We wanted to get feedback on our work since we were uncertain if the the tests we made did cover the two comments on the JIRA issue tracker. Another uncertainty was if all the tests from the old test suite were now covered by JUnit tests. If they were the old harness test could be removed.

After a couple of days with no feedback we decided to withdraw from the project. The reason for this was lack of time. We had encountered some obstacles during our intervention, and this was our main goal.

One week after withdrawing from the project we got some feedback on the patch submitted. Even though we had withdrawn from the project we used one day to complete the patch. This patch is now committed.

6.4.4 Evaluation

This intervention was more focused than the second one. We worked on a specific task and had a clear and well defined goal to work with. Involving the community and letting them know what we were working on made the whole process more efficient. By involving them in the action planning and letting them see our progress step by step we gave them the opportunity to give feedback and share their knowledge with us.

If we were more experienced with JUnit and the specific part of the Java API, which we had not seen before, this iteration would have been even swifter. It is worth noting that part of the motivational factor for some of the hobbyist OSS developers is to learn something new and enjoy it. We did indeed learn something new and had fun while doing it.

6.4.5 Reflection

Again we found that documentation is very important for a Newcomer. We had to go through a lot of Wiki pages and API documents to get familiarized with JUnit before we could start the actual programming. The documentation we did find was of good quality, but the amount found can be quite overwhelming. Adding links to the the different JUnit pages at the end of the IntroToJUnit page

is an easy way to help a Newcomer find the information he is looking for. The links should contain a short description of the info found one the linked page. Now only a few pages and some external resources are linked from the document most Newcomers will turn to when starting to work on JUnit. Adding links to the Wiki pages should then not be a hard task to do. A Newcomer can of course ask on the mailing lists were to find this information, and the response time on such a question is, in our experience, quite rapid. The problem here will be the unnecessary noise this creates on the mailing lists. If the Developers have to start spending more and more time answering these kinds of questions, they will probably become tired of it. As a consequence their response time will increase while the politeness will decrease.

We found the folder structure to be confusing when working on the JUnit tests. The folders might have a good system and be the way it is for a good reason, but this reason is not written in any documentation we could find. This makes it hard to navigate through the folders when looking for specific files. Our experience from other projects is that they have a Readme file in the different sub folders, explaining the directory structures. Using these makes it easy to understand were to look for information, and to understand the structure and modularity of the code. We did find some information when building and reading the java-doc, but this was hard to reach and lacking. This makes the information unpractical for a Joiner.

When we started converting the old harness tests to JUnit we found our own programming experience in this area to be insufficient. We had to spend some time reading the Java API and refreshing our own skills before we could start coding. This is an obstacle that got to do with personal attributes, not the Apache Derby project. Good documentation can help the Newcomers find the right information, but it is not the projects job to teach the Newcomers programming. Therefore we choose not to look further into this obstacle.

We chose to convert one of the old harness tests suggested to us on the mailing lists. When reading the JIRA issue we found that most of the tests in the old harness test suite were already covered by a JUnit test. This greatly reduced the workload of converting the test to JUnit. Not because the programming became easier, but because we did not have to spend so much time writing the tests. Learning about JUnit programming etcetera still takes the same amount of time, but we did not have to write all the lines of code needed for all the test cases. This enabled us to finish writing this test within a reasonable amount of time. A volunteer can usually not spend as much time as a paid programmer working on a project. This unable a volunteer to assign himself to issues that are too time-consuming. To mitigate this one can make use of the field on the JIRA issue tracker were the reporter can suggest an estimated time to finish an issue. We all know that time estimates on programming tasks are hard to do. This would work if the Developers reporting the issues, that would need little

time to do, resisted the urge to just do them themselves. Leaving small issues of minor importance to Newcomers could lower the Contribution Barrier. This would allow Contributors with little time, to contribute.

Part III

Discussion and Conclusions

Chapter 7

Discussion

In this chapter we go through the various aspects of this report. We start by discussing our results and group these according to the theory presented in chapter 5. We then continue to criticise the the research questions and research method.

7.1 Results

In this section we present our results and discuss their rigor and relevance. We link them to the themes from our literature study in chapter 4. In table 7.1 the obstacles from our iterations are presented. These obstacles are discussed throughout this section.

It.	Ob.	Description	Suggested Fix
1	O1	IRC	-
2	O2	$CLASSPATH	a) Description b) Cut and paste to text file
	O3	Outdated documentation	Person responsible for documentation effort
	O4	Test log	Guide Newcomers to JUnit issues
	O5	Wrong issue flags	-
3	O6	Scattered information	Collection of link
	O7	Folder structure	Readme file describing the folder structure in sub folders
	O8	Coding experience in specific area	-
	O9	Time consuming issues	Give time estimate

Table 7.1: Obstacles from Our Iterations.

7.1.1 Joining-script

In section 5.3 we concluded that the availability of a Joining-script was not an obstacle in Apache Derby, and that we thus would not look further into this issue. This conclusion is still valid, but now that our knowledge about the project has increased, we see that the quality of the Joining-script could have been investigated. We based our evaluation of their Joining-script largely on our own experience when working with it last semester. When going through the Joining-script this semester looking at the old harness suite, we found that this suite was far from an ideal place to start for a Newcomer. So even if the Joining-script existed and was easy to find, it could have been updated to reflect the focus on JUnit instead of the old harness test suites.

We did encounter all our obstacles following the Joining-script, but this does not mean that all the obstacles are a consequence of it. The ForNewDevelopers document should be redone to make sure that Newcomers get guided to JUnit testing rather than the old harness test suite. If a Newcomer follow the Joining-script they describe, a Newcomer will get a good start with the community, in the way a Joining-script is meant to do.

7.1.2 Motivation

The Apache organisation is a well oiled machinery with a lot of experience running Open Source projects. Their projects are well known, and they have a lot of lurkers overlooking the progress of those projects. Feedback is fast and constructive, and Joiners are allowed to ask questions without being ridiculed by more experienced Developers. All of this means that the visible factors contributing to motivating Joiners and Newcomers are present. However, we found other obstacles that can influence a persons motivation to continue in a project.

O4 and O5 are closely knit together. When we encountered those obstacles we worked on an issue that we thought would be quite easy to solve. It was flagged as Newcomer and proposed on the ForNewDevelopers web page. Instead of being able to solve it we ran into a huge incomprehensible test log. We have worked on other issues where we easily asked questions, but we did not want to ask for assistance relating to this issue since it would discredit us publicly on the mailing list. We did not want the Developers to know that we could not handle a Newcomer issue on our own, since they are the same people that would evaluate our work later on. This illustrates that even if everything is in place to help and motivate Newcomers, the use of tools like the JIRA issue tracker and personal resources have to be correct. In this case the issue was wrongly flagged, and it was discouraging to us that we needed help to solve it.

7.1.3 Contribution Barrier

As mentioned in section 4.2.3, a Contribution Barrier is any reason keeping a Newcomer from contributing to the project, but with a higher abstraction level than an obstacle. Contribution Barriers can be personal. This means that they can be eased by the person, but not the project. An example of this is O8, were our own lack of programming experience in the JUnit arena is the obstacle. Good documentation can help smooth out this obstacle, but in the end only experience will help the Newcomer. We have therefore ignored this kind of Contribution Barriers.

The Contribution Barriers we found while working on the project are listed in table 7.2.

CB	Name	Description
CB1	Utilities	(O1) The informal information channel IRC is not used to its full potential. (O5) JIRA issues must have the correct flag.
CB2	Documentation	(O2, O3 and O6) Scattered/Outdated documentation
CB3	Time	(O9) Time consuming issues makes it hard for free agents to join
CB4	Meta data	(O4) Test logs in the old harness suite is too hard to read for a Newcomer.(O7) The folder structure of Apache Derby source code is not self-explanatory.

Table 7.2: Contribution Barriers.

CB1

The Apache foundation makes sure the utilities needed to run a project are in place. However, it is up to the project to use these utilities in the best manner possible. The full potential of IRC is not used. The activity did increase when the students from GSoC joined the project, but it is hard to tell what will happen if the students stop asking questions. Normally there is only one to three experienced Developers in the #Derby channel, and they seldom answer questions not asked directly to them.

The JIRA issue tracker is a great utility to control the tasks needed to be done in the project. All issues pass through this utility. This result in a lot of JIRA issues being created. The handling of these issues are important. Slipping with flagging of difficulty or importance of tasks can be misleading to other Developers wanting to help out with an issue.

CB2

We found multiple obstacles that had to do with the documentation of Derby. O2 is an obstacle because there was a mistake in the example on how to set the $CLASSPATH. This might seem trivial, but to a Newcomer that lacks the deeper understanding of how this variable works, a correct example is needed to be able to progress in the project. We also found outdated documentation in O3 and the documentation to be very scattered in O6.

When searching for information on JUnit testing, you will find loads of different pages giving tips and help with single issues and cases, but not a document explaining how to tie this information together. You have to guess from the index name of the Internet page what it will contain, and what help it might give you. Many pages got good, explanatory names, but not all.

CB3

Time is an important criteria when working on anything, specially for free agents in OSSD. If a Newcomer is to work on something in his/her spare time knowing the time limit of the task is very useful. This again (as O5 also focus on) shows the importance of competent use of the issue tracker to give a good indication of the workload an issue results in. Payed Developers will not be as affected by the time constrains as free agents.

CB4

By Meta Data we mean "data about data". The lack of information on how to interpret the test logs from the old harness test suite (O4) is an obstacle. When a Newcomer to the project gets a directory containing 486 folders when running a test suite, with no further information on what they contain, he is bound to get confused.

Browsing the directory structure of the Apache Derby source code is not self-explanatory(O7). The folders seen when first opening the source code looks good, and they are self explanatory. But as soon as one goes further into the folder structure, it gets harder. One will meet a total of 2747 folders containing 12905 files, but not a single file explaining how the folders are structured. An example is when we tried to find the JUnit tests. We started by opening the Testing folder. Then continued past the mandatory /org and /apache folders before moving on to /derbytesting[1]. Here we found a JUnit folder, and we thought jackpot at first try. But when we opened this folder we did not find the tests, but super-classes made to help produce new JUnit tests. After some more searching we ended up in another test folder that also contained another JUnit folder[2]. But again we

[1] $DERBY_SOURCE/java/testing/org/apache/derbyTesting/junit/
[2] $DERBY_SOURCE/java/testing/org/apache/derbyTesting/functionTests/tests/

were confused, because we could only find a few of the JUnit tests here. After checking all the sub folders of the /tests folder, we found that the JUnit tests were scattered between all of them. To us it seems that there was no greater plan to where the files were put, and that it is up to the individual committer do decide were he thinks a file belongs.

7.1.4 Artifacts

The Apache Foundation got long experience with OSSD projects. They make sure the artifacts needed for a project to be successful are in place. They help facilitate web pages, Wiki, JIRA issue tracker, IRC channel, mailing lists, legal help et cetera. A problem we found when using all these artifacts was that the information on them could vary. An example can be found below, were we show an e-mail that was sent on the developer list 03.06.2008.

```
**************************************************************

Hello,

I observe that the lists of committers in the STATUS file and in Jira
are out of sync. There are names in the STATUS file that are not in
the Jira committer group, and vice versa.

I have not found a definite authority listing committers, but I know
the access control structure for Subversion would give us the current
situation. Digging out votes from the mail archives would also work,
but requires some more work.

Do we generally want to keep the various lists in sync?

-Developer

**************************************************************
```

When a project got the amount of artifacts found in Apache Derby it is very hard to keep everything up to date. Problems that arise because of this can be observed in most of the artifacts in Apache Derby. Examples of this are the issue tracker and the documentation efforts that were discussed in 7.1.3.

The artifact that will not get clouded or outdated because of the use of other artifacts is IRC. In our first iteration we tried to get more people to join IRC. We started logging the #Derby channel. After the first iteration we decided to keep on logging, to see if the increased activity we saw would continue throughout the semester. We have used the IRC log to create a chart of the activity per day

on the channel, and the result can be found in 7.1. As you can see the activity increased significantly when the GSoC students joined the channel. March 24 is the first day that the activity significantly increased. This was two days after the first GSoC applicants had said hello on the mailing list. This is when we started our first iteration. However, our intervention did not start before June 07. The goal of the intervention was to increase the IRC activity. It is hard to evaluate the results of this intervention because the activity had already increased when we started it. Up until March 23 there were an average of 13 lines spoken a day. From March 24 the average number of lines spoken per day had increased to 78.

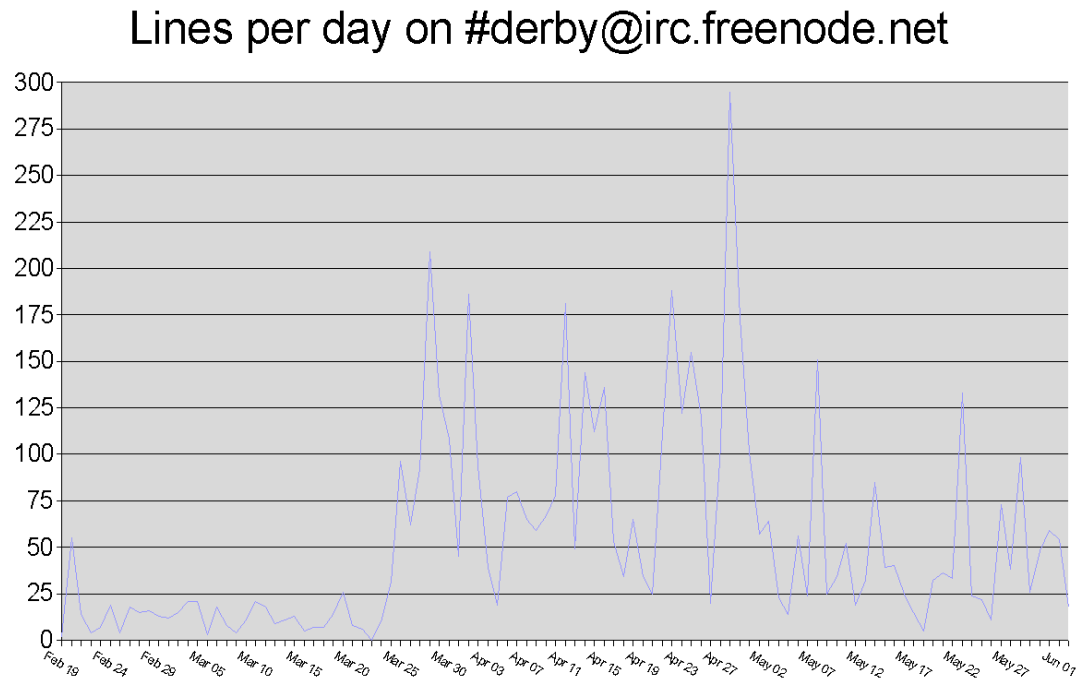

Figure 7.1: Derby IRC Activity Chart.

In figure 7.2 we grouped the IRC users according to their roles in the project. As can be seen on the pie chart, there are three Developers particularly active on IRC, but only one that really distinguish herself, namely Developer1. This is the same person that was the facilitator for GSoC. Developer2 got more than twice the activity of Developer3. This Developer was a Co-mentor for GSoC. The remaining big groups are the accepted and declined GSoC applicants, Others and one user that was exceptionally active.

The groups Other dev, Others, User1 and Developer3 are not talking about issues related to GSoC. These groups make up for a total of 35.44% of the activity, or an average of 28 lines pr day. This is an increase from the 13 lines a day we saw before the GSoC activity started. We think this increase in other activity

was a result of the additional activity from GSoC. The activity this project has brought to IRC has made it a more interesting place to be.

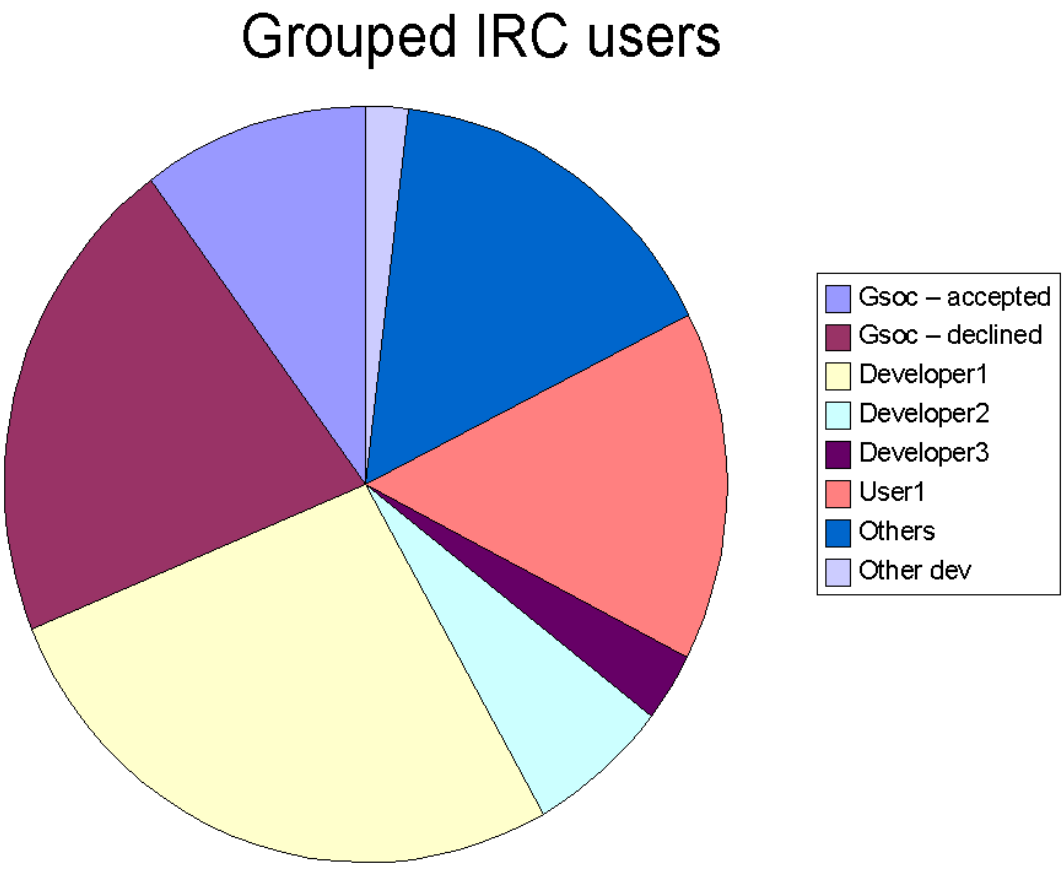

Figure 7.2: Grouped IRC Users.

7.2 Research Questions

In this section we defend our choice of research questions. A possible bias with our answers to the researcher questions is that we were actively seeking out obstacles.

We did not change these research questions throughout our project. The nature of the RQs makes them hard to change. We did not identify any additional obstacles in our first iteration, but still we kept our RQs. The RQs can be viewed as goals for our research, and this is both a weakness and a strength with them. It is easy to stay focused and know what the goal is when the research questions remain unchanged. On the other hand it, if the research questions have flaws or

turn out to be uninteresting they should be open for change. We believe that we would have changed the RQs if we were getting no results.

7.3 Research Method

In this section we discuss the five principles from [Davison et al., 2004]. First we discuss whether or not our research followed the principle. Second we suggest how to archive fulfillment and present possible refinements to CAR.

7.3.1 Principle of the Researchers-Client Agreement

In [Holum and Løvland, 2007] we did not have a RCA and that turned out to be problematic. Learning form our previous research we started this project by contacting a possible client as mentioned in section 5.1. During the meeting with the client we got suggestions on which areas we should focus our research on. We never got the chance to present the research method to the client and the suggested contract was never signed. This resulted in us having a focus area given by the client, but since the client never committed to the RCA they were not included throughout our research. When taking this into consideration, this report does not fulfill the principle of the RCA.

CAR is not specifically aimed at doing research on OSSD. We believe changes to the RCA principle can make it easier to apply the method in future OSSD studies, while at the same time keeping the intended rigor and relevance. We suggest to post the agreement publicly on a suitable mailing list stating the intentions of the research and how the community might benefit from it. Instead of having a signed agreement the community can be said to commit to the agreement if they reach consensus and accept the RCA. This would work in an Apache community where they have processes to reach consensus. Other OSSD-projects might have one leader. This leader would then have veto rights when voting. Similar suggestions could be made for a variety of OSSD organisational structures. The bottom line is to reach an agreement with those who control the project.

7.3.2 Principle of the Cyclical Process Model

Our interventions were clearly build up around the CPM and this is reflected in the structure of the report. We did our research in an iterative manner with the output of each phase used as the input for the next. This helped us accomplish structured and focused work. The evaluation phase was originally intended to include a client, but we had no client to discuss an iteration with. Instead we evaluated the data collected from the iterations ourselves.

We sometimes made the mistake of mixing evaluation and reflection when we verbally discussed the actions done in an iteration. We ended up with evaluating the actions done in an intervention and presenting the collected data in an evaluation phase. In a reflection phase we use the data previously presented and knit it to our research questions.

7.3.3 Principle of Theory

In the pre-study we presented literature around OSS and Joining. This literature was then applied to Apache Derby. The theory did not strictly inform our diagnose phases but our research questions were strongly influenced by it. Our research questions are by their nature resistant to change when used with CAR. They are the basis for all our diagnoses and action plans. By this all our interventions are indirectly influenced by theory through the research questions. Theory is also used above when discussing the results. We conclude by this that our research did fulfill the principle of theory.

We do not have any tuning of this principle when it comes to applying CAR when studying OSSD. It is worth noting that if a future study has research questions unlike ours, new theory might need to be introduced in succeeding diagnosis. Introducing new theory and presenting it to the client takes time. This master's thesis project has a 20 week duration limit. In our experience one would need a lot more time to see the effects of the actions taken and to be able to measure the impact of the theory introduced. A project spanning 2-3 years would be more reasonable.

7.3.4 Principle of Change Through Action

This is by far the most challenging principle to achieve when studying OSSD. To fulfill this one should have a fully committed community ready to accept possibly radical changes to see how they work. A RCA must exist and the community must be devoted to it. Since we had no signed RCA we did not have a devoted community, because they had nothing to be devoted to. However our attempt to increase the activity on IRC was successful and that is "a change through action". By this we conclude that we partly fulfilled this principle.

We do not have any suggestions on how future research can fulfill this principle without a RCA. Our suggestion is that a RCA must be in place. Given that a RCA is a though challenge to achieve with a OSS community leads us to the conclusion that if there is no indication of a RCA being achieved prior to the first intervention the CAR method should be discarded and replaced, possibly by an ethnographic study. As a side note, our effort to ease Joining is highly dependant on a frequent supply of Joiners to measure the impact of the changes.

7.3.5 Principle of Learning Through Reflection

Since we lacked an explicit client we did not make any formal progress reports about our research while performing it. But, we kept a diary where we maintained an informal progress report and wrote down lessons learned. Our implicit client, the Apache Derby community, got progress reports through discussions on the mailing list, IRC and JIRA issue tracker. In the reflection phases we did not involve the Apache Derby community, but reflected on the output from the evaluation phases. Our goal was to find obstacles which we did by completing our Joining. And findings concerning our research questions are based on our own experience as well as observations of other Newcomers. The other Newcomers were not actively included in the reflection phases, but their generated output were. We as researcher have reflected on the results of this project, but have not included a client in the process. Again we partly fulfill the principle.

To fulfill this principle when studying OSSD and having the OSS community effectively working as the client, one should include the affected members of the community in the reflection phase. By doing that one would get acknowledgement that the results are sound. The researchers would then get the needed feedback from the community necessary to fulfill this principle.

Chapter 8

Conclusions and Further Work

In this chapter we present the answers to our research questions, before we present our refined model for CAR and further work. The research questions were:

- RQ1: Which obstacles are encountered by Newcomers to Apache Derby when Joining?
- RQ2: What can be done to ease the Joining process?

8.1 Conclusions

Research question 1 is a direct result of our conversation with a company working on Apache Derby. They wanted external forces looking at Joining trying to find obstacles. The obstacles encountered during our Joining can be found in table 8.1 which is the answer to RQ1.

Ob.	**Obstacle Description**	**Ob.**	**Obstacle Description**
O1	IRC	O6	Scattered information
O2	$CLASSPATH	O7	Folder structure
O3	Outdated documentation	O8	Coding experience in a specific area
O4	Test log	O9	Time consuming issues
O5	Wrong issue flags	-	-

Table 8.1: Final Obstacle List.

Research question 2 is the natural next step from RQ1. Identifying the obstacles, but not trying to improve them, and thus easing the Joining, would not make much sense. We group the obstacles found from RQ1 into Contribution Barriers in table 7.2. To answer RQ2 we look further into these Contribution

Barriers and present possible solutions for how to mitigate the effects they have for a Newcomer to the project.

Contribution Barrier 1 concerns the use of **utilities** in an Open Source project. It is important for a Joiner to quickly get familiarized with the available utilities of a project. Without these it will be extremely hard to stay updated on what is going on in the project. The availability on IRC by experienced personnel is very important for a Joiner who needs to have an informal conversation before exposing himself to the community. Also the use of the JIRA issue tracker is important to a Joiner. Much of the communication in a project, and all the issue assignments happens on the JIRA. It is therefore very important that the flags on the JIRA issue tracker is correct and reflect the actual knowledge level needed to solve a task.

Contribution Barrier 2 raises the very important issue of **documentation**. This is not an issue solely for OSSD projects, but because the tacit information is hard to share in an OSSD project, documentation is very important. The amount of documentation in an OSSD project makes it hard to have full control. For a person in the Joining the most important document to lean on at the start will be a Joining-script. Keeping the Joining-script updated and easily accessible is something any OSSD project should strive to do.

Contribution Barrier 3, the **time** issue, can be viewed as a personal contribution barrier in the same way as we choose to look at O9, as a personal contribution barrier. This barrier can however be mitigated by the OSSD projects with much more ease than solving O9. Making sure that there are small issues available for Newcomers with little time on their hands would help overcome this barrier. It might be easier for an experienced Developer to solve these issues themselves as soon as they are found, but by letting Newcomers solve these they attract new people that get a positive first contact with the community. On small simple issues there should not be a big problem adding a time factor to the task as well. This way the Newcomer will know that he can handle the workload before assigning himself to a task.

Contribution Barrier 4 focuses on the use of **meta data** in the OSSD projects. It is not trivial to get familiar with the code base in a large, mature project. Several meta datas should be in place to help a Newcomer. First off good open software architecture diagrams describing the code are needed. Secondly you need Readme files in the directory structure of the source code explaining the use of the folder and its sub folders. The last mitigating factor we found was good meta data on how to read the test logs. Folders and output files are not trivial to find for a Newcomer, and meta data explaining what the files and folders contain is essential for the Newcomers experience.

8.2 Refined Research method

The absence of an explicit client consequently restricted us from fully fulfilling the principles of CAR. Below is a summary of our suggested refinements to the research method when it is used to study OSSD.

- RCA
  - Be open about doing research.
  - Propose a RCA to the community.
  - Do not continue without an agreed upon RCA.
- CPM
  - Actively include the community in the different phases.
  - Evaluation is to check if the actions were correct and evaluate the data collected.
  - Reflection is to use the data to answer the research questions.
- Theory
  - Present the used theory to the community.
- Change Through Reflection
  - Impossible without a RCA.
  - Some research areas might have unforeseen dependencies.
- Learning Through Reflection
  - Report lessons learned to the community.
  - Use feedback from the community.

CAR without modifications does not work on a short term student project. Students do not have the manpower to *read* all the literature needed to back up all the phases while at the same time *use* this literature actively for action planning, intervention, evaluation and reflection. You will need 4-10 people over a period of 2-3 years, working on a project with a client that is willing to introduce the changes suggested.

8.3 Further work

It would be interesting to implement our suggestions on how to ease the Contribution Barriers in the Apache Derby Community and try to measure the impact. This can be done as a new iteration.

Using the same research questions while looking at other OSSD projects to extrapolate new results is an exciting possibility. This could verify or disprove the claims of this thesis.

The research questions did not change while writing this thesis. Changing the the RQs in a succeeding study to focus more spesifically on one of the following; Joining-script, Motivation, Contribution Barrier or Artifacts could give a much deeper insight into the mentioned aspects.

While doing any of the suggested future work, trying out the refined research model should be considered.

Bibliography

[Avison et al., 1999] Avison, D. E., Lau, F., Myers, M. D., and Nielsen, P. A. (1999). Action research. *Commun. ACM*, 42(1):94–97.

[Baskerville and Wood-Harper, 1996] Baskerville, R. L. and Wood-Harper, A. T. (1996). A critical perspective on action research as a method for information systems research. *Journal of Information Technology*, 11(3):235 –.

[Beard and Kim, 2007] Beard, A. and Kim, H. (2007). A survey on open source software licenses: student paper. *J. Comput. Small Coll.*, 22(4):205–211.

[Bird et al., 2007] Bird, C., Gourley, A., Devanbu, P., Swaminathan, A., and Hsu, G. (2007). Open borders? immigration in open source projects. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 6, Washington, DC, USA. IEEE Computer Society.

[Crowston et al., 2007] Crowston, K., Li, Q., Wei, K., Eseryel, U. Y., and Howison, J. (2007). Self-organization of teams for free/libre open source software development. *Information and Software Technology*, 49(6):564–575.

[Cuéllar, 2005] Cuéllar, L. E. (2005). Open source license alternatives for software applications: is it a solution to stop software piracy? In *ACM-SE 43: Proceedings of the 43rd annual Southeast regional conference*, pages 269–274, New York, NY, USA. ACM.

[Davison et al., 2004] Davison, M., Martinsons, G., and Kock, N. (2004). Principles of canonical action research. *Info Systems Journal*, 22(14).

[Fitzgerald, 2006] Fitzgerald, B. (2006). The transformation of open source software. *Forthcoming in MIS Quarterly*, 30(3).

[Gacek and Arief, 2004] Gacek, C. and Arief, B. (Jan-Feb 2004). The many meanings of open source. *Software, IEEE*, 21(1):34–40.

[Hars and Ou, 2002] Hars, A. and Ou, S. (2002). Working for free? motivations for participating in open-source projects. *Int. J. Electron. Commerce*, 6(3):25–39.

[Herraiz et al., 2006] Herraiz, I., Robles, G., Amor, J. J., Romera, T., and Barahona, J. M. G. (2006). The processes of joining in global distributed software projects. In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 27–33, New York, NY, USA. ACM.

[Hertel et al., 2003] Hertel, G., Niedner, S., and Herrmann, S. (2003). Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research Policy*, 32(7):1159–1177.

[Holum and Løvland, 2007] Holum, H. and Løvland, S. E. R. (2007). Action research: A study of oss testing in apache derby. Master's thesis, NTNU.

[Iversen et al., 2004] Iversen, J. H., Mathiassen, L., and Nielsen, P. A. (2004). Managing risk in software process improvement: An action research approach. *MIS Quarterly*, 23(3).

[Jensen and Scacchi, 2007] Jensen, C. and Scacchi, W. (2007). Role migration and advancement processes in ossd projects: A comparative case study. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 364–374, Washington, DC, USA. IEEE Computer Society.

[Li et al., 2005] Li, J., Conradi, R., Slyngstad, O., Bunse, C., Khan, U., Torchiano, M., and Morisio, M. (19-22 Sept. 2005). Validation of new theses on off-the-shelf component based development. *Software Metrics, 2005. 11th IEEE International Symposium*, pages 26–26.

[Lovgren and Racer, 2000] Lovgren, R. H. and Racer, M. J. (2000). Group dynamics in projects: Don't forget the social aspects. *Journal of Professional Issues in Engineering Education and Practice*, 126(4):156–165.

[McKay and Marshall, 2001] McKay, J. and Marshall, P. (2001). The dual imperatives of action research. *Information Technology and People*, 14.

[Mockus et al., 2002] Mockus, A., Fielding, R., and Herbsleb, J. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346.

[Rapoport, 1970] Rapoport, R. (1970). Three dilemmas in action research. *Human Relations*, 23(4):499 – 513.

[Raymond, 2001a] Raymond, E. S. (2001a). A brief history of hackerdom. *The Cathedral and the Bazaar*, 2(1).

[Raymond, 2001b] Raymond, E. S. (2001b). The cathedral and the bazaar. *The Cathedral and the Bazaar*, 2(1).

[Scacchi, 2007] Scacchi, W. (2007). Free/open source software development: recent research results and emerging opportunities. In *ESEC-FSE companion '07: The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering*, pages 459–468, New York, NY, USA. ACM.

[Østerlie and Jaccheri, 2007] Østerlie, T. and Jaccheri, L. (2007). A critical review of software enginering research on open source software development. In *2nd AIS SIGSAND European Symposium on Systems Analysis and Design*.

[Susmanand and Evered, 1978] Susmanand, G. I. and Evered, R. D. (1978). An assessment of the scientific merits of action research. *Administrative Science Quarterly*, 23(4):582–603.

[Tilly, 1999] Tilly, C. (1999). *Durable Inequality*. University of California Press, Berkley, CA.

[Trist, 1976] Trist, E. (1976). *Exsperimenting with Organizational Life: The Action Research Approach*. Plenum Press, NY, NY.

[von Krogh et al., 2003] von Krogh, G., Spaeth, S., and Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(1):1217–1241.

[Wang et al., 2005] Wang, F.-R., He, D., and Chen, J. (18-21 Aug. 2005). Motivations of individuals and firms participating in open source community. *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, 1:309–314.

[Wheeler, 2007] Wheeler, D. A. (2007). Why open source software / free software (oss/fs, floss, or foss)? look at the numbers! *www.dwheeler.com*.

Part IV

Appendices

Appendix A

Abbreviations

AR	Action Research
ASF	Apache Software Foundation
CAR	Canonical Action Research
CPM	Cyclical Process model
FOSS	Free and Open Source Software
FLOSS	Free/Libre Open Source Software
FSF	Free Software Foundation
GNU	GNU's not Unix
GPL	General Public Licence
GSoC	Google Summer of Code
IRC	Internet Relay Chat
IT	Information Technology
MIT	Massachusetts Institute of Technology
OSS	Open Source Software
OSSD	Open Source Software Development
RCA	Researchers-Client Agreement
RQ	Research Question
SVN	Subversion

Appendix B

JUnit URLs

These are some of the URLs used in the iteration when converting a test from the old harness suite to JUnit. Last accessed 26.05.2008.

http://wiki.apache.org/db-derby/ConvertOldTestToJunitTips
http://wiki.apache.org/db-derby/DerbyJunitTestConfiguration
http://wiki.apache.org/db-derby/DerbyJUnitTesting
http://wiki.apache.org/db-derby/DerbyTopLevelJunitTests
http://wiki.apache.org/db-derby/IntroToJUnit
http://wiki.apache.org/db-derby/JunitAssertMessages
http://wiki.apache.org/db-derby/KillDerbyTestHarness
http://mail-archives.apache.org/mod_mbox/db-derby-dev/200607.mbox/%3c44BFD7EE.3020708@apache.org%3e
http://svn.apache.org/viewvc/db/derby/code/trunk/java/testing/README.htm?view=co

Appendix C

Researchers-Client Agreement

Researcher Client Agreement

This is a contract between Svein Erik Reknes Løvland, Henrik Holum and Sun Microsystems. This contract is valid from _____ to _____.

Research Method: CAR

We agree that Canonical Action Research is the appropriate research method for this project.

Focus: Joining

The research will focus on Joining, which is the process of transforming a newcomer to an active developer.

Commitments: Feedback

There will be at least one meeting with the client each evaluation phase. There will be a minimum of two evaluation phases in this project. Meetings outside of the evaluation phase might be needed, but this will be agreed upon with the client if the situation occurs. A notice will be sent at least 48 hours before the time of the meeting. E-mails will be answered within 24 hours, or withing the end of the next working day. The researchers will give the contact a status report once a month, and within 24 hours if requested.

Roles:

The client will in a working situation not treat the researcher any different than they would treat other participants in Apache Derby. The researchers will participate in Apache Derby and will take a role as newcomers. The only time the client will treat the researchers different from other developers will be during the feedback meetings in the evaluation phase. The researchers will get a contact in SUN with whom they will be communicating with regards to meetings and feedback.

Goal: Improve the Joining process

The goal of the project is to remove obstacles which are encountered during our Joining process. Feedback on the intervention phase will be needed by the client to asses any progress in eliminating these obstacles. The time it takes and the number of obstacles encountered by newcomers when compiling and running the DerbyAll test suite will be used to evaluate the measures taken.

Data gathering and analysis:

Data will be collected from the researchers interaction with the community and analyzed with suitable methods. Because we do not know exactly what data that will be collected yet, the exact gathering method and analyze method will be agreed upon through dialog at a later stage of the project.

_____	_____	_____	_____
Sun Representative	Henrik Holum	Svein Erik Reknes Løvland	Date, Place