

PROJECT CS  
UPPSALA UNIVERSITY

LoPEC  
LOW POWER ERLANG-BASED CLUSTER

---

# Project Report

---

Fredrik Andersson  
Axel Andrén  
Niclas Axelsson  
Fabian Bergström  
Björn Dahlman

Christofer Ferm  
Henrik Nordh  
Vasilij Savin  
Gustav Simonsson  
Henrik Thalin

January 14, 2010

## Abstract

This document describes the project methodology and development process applied during the project which resulted in the LoPEC cluster system. The project was carried out by ten computer science students at Uppsala University during the autumn of 2009. In collaboration with ***Erlang Solutions*** and Streamfile with the initial purpose of developing a power-efficient GPGPU cluster. The focus of this document is the team's reflections on the entire development of the system.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Methodology</b>	<b>4</b>
2.1	Scrum . . . . .	4
2.2	Theory . . . . .	5
2.2.1	Scrum Roles . . . . .	5
2.2.2	Sprint Planning Meeting . . . . .	5
2.2.3	Daily Scrum Meeting . . . . .	6
2.2.4	Sprint Review Meeting . . . . .	6
2.2.5	Retrospective . . . . .	6
2.3	How we did it . . . . .	6
2.3.1	Scrum Roles . . . . .	6
2.3.2	Sprint Planning Meeting . . . . .	7
2.3.3	Daily Scrum Meeting . . . . .	7
2.3.4	Sprint Review Meeting . . . . .	7
2.3.5	Retrospective . . . . .	8
<b>3</b>	<b>Milestones</b>	<b>9</b>
3.1	Sprint 1 . . . . .	9
3.2	Sprint 2 . . . . .	9
3.3	Sprint 3 . . . . .	9
3.4	Erlang User Conference . . . . .	10
3.5	Sprint 4 . . . . .	10
3.6	Sprint 5 . . . . .	10
3.7	External Review . . . . .	10
3.8	Final Presentation . . . . .	10
<b>4</b>	<b>Resources</b>	<b>11</b>
4.1	The Team . . . . .	11
4.2	Hardware . . . . .	12
4.3	Training . . . . .	12
4.3.1	Course in Erlang . . . . .	12
4.3.2	Course in Scrum . . . . .	12

4.4	Tools . . . . .	12
4.4.1	Communication . . . . .	12
4.4.2	Version Control . . . . .	13
4.4.3	Issue Tracking and Continuous Integration . . . . .	13
4.4.4	Development environment . . . . .	13
4.4.5	Code Testing . . . . .	14
<b>5</b>	<b>Problems</b>	<b>15</b>
5.1	Not Being Proficient with Erlang . . . . .	15
5.2	Nonexisting Backlog . . . . .	15
5.3	No Prior Experience with Clusters . . . . .	15
5.4	Task Division . . . . .	16
5.5	Poor Testing and Continuous Integration Experience . . . . .	16
5.6	Ignoring the Work Hours . . . . .	16
5.7	Issues in the Working Environment . . . . .	17
5.8	Incommunicado . . . . .	17
5.9	Scrum Master . . . . .	17
<b>6</b>	<b>What we learned</b>	<b>19</b>
6.1	Development Process . . . . .	19
6.2	Project Rules . . . . .	20
<b>7</b>	<b>Conclusion</b>	<b>21</b>
<b>A</b>	<b>Individual Contributions</b>	<b>24</b>

# Nomenclature

**Backlog item** A project feature desired by customer

**Erlang Solutions** Former company name - Erlang Training & Consulting

**Erlang Solutions** Formerly Erlang Training & Consulting

**Erlang** A functional programming language developed by Ericsson

**GPGPU** General-purpose computing on graphical processing units

**MoSCoW** Must, Should, Could, Would - A system to prioritize things in order of importance.

**OTP** Open Telecom Platform, a collection of modules for Erlang

**OpenCL** A framework for writing programs that execute on a multitude of processors such as CPUs and GPUs

**Product backlog** A prioritized list of all the desired features of the product being developed, put together by the product owner.

**Scrum** An iterative incremental framework for managing complex work, such as product development

**Sprint backlog** A list of tasks that should be completed during the current sprint, based on features picked out from the product backlog.

**Sprint** A period of time during which the team works on agreed on tasks. The recommended length is 2 to 4 weeks

**Tasks** Specific developments required to implement a feature

**VCS** Version Control System

# Chapter 1

## Introduction

The purpose of this project was to develop a low-power *GPGPU* cluster using off-the-shelf hardware. Our timespan was approximately three months, and the main idea was to utilize *Erlang* and *OpenCL* to implement the system, and measure how well it would fare. Our team consisted of ten Computer Science students from Uppsala University, who developed the product using the *Scrum* process.

Two main goals for the course, according to the official course description, was “give insights into how a big project is run (from planning to realization), how to construct a complex distributed system”. The first goal became a reality simply by the size of the project (ten students) in combination with the use of Scrum, which included extensive planning and structuring of work before any work commenced. While we in many ways did not have defined requirements of the cluster early on, the cluster quickly evolved towards the second goal, as the distributed property of the cluster became a central part of its design.

## Chapter 2

# Methodology

### 2.1 Scrum

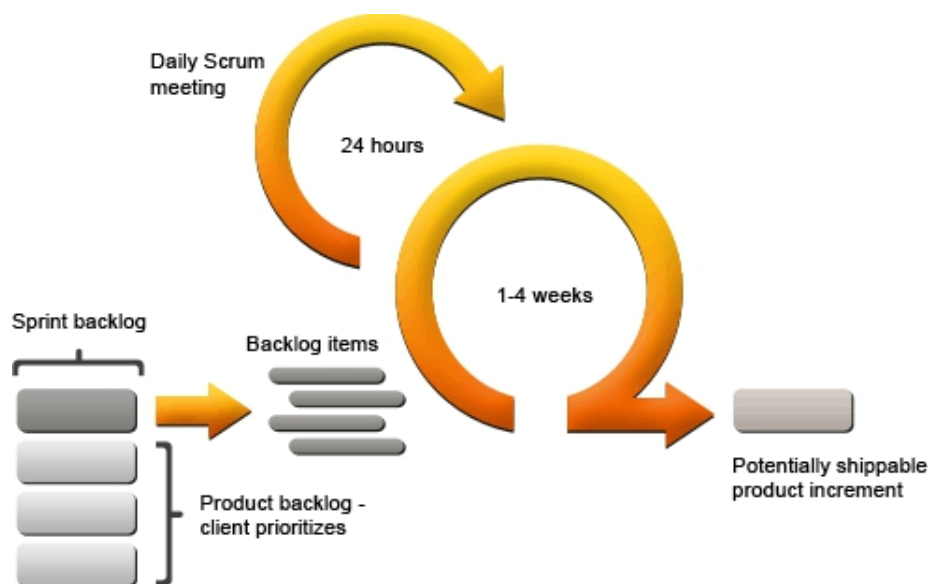


Figure 2.1: Scrum diagram

Scrum is an agile software development process, best suited for projects with rapidly changing requirements.

This chapter briefly covers the scrum theory <sup>1 2 3</sup> and how we used Scrum ourselves in the project.

---

<sup>1</sup><http://www.controlchaos.com/about/>

<sup>2</sup><http://www.scrumalliance.org/>

<sup>3</sup><http://www.mountangoatsoftware.com/scrum/overview>

## 2.2 Theory

### 2.2.1 Scrum Roles

**Product Owner** - acts as a spokesman for the customer, and defines features of the product based on each *Backlog item* or each specific request of the customer. He should prioritize these features according to the market value, decide on a release date for the product, and is responsible for the profitability of the product. The product owner should also adjust the contents of the features and their priority after every *Sprint*, and decide if what has been produced is acceptable.

**Scrum Master** - responsible for making sure a Scrum team lives by the values and practices of Scrum, and for removing any impediments to the progress of the team. As such, he should shield the team from external interferences, and ensure that the Scrum process is followed, including issuing invitations to the daily Scrum meetings (See 2.2.3), sprint reviews (See 2.2.4) and the sprint planning meetings (See 2.2.2).

**Scrum Team** - the group of people developing the product. There is no personal responsibility in Scrum, the whole team fails or succeeds as a single entity.

### 2.2.2 Sprint Planning Meeting

The sprint planning meeting is attended by the product owner, the Scrum master, the team and any interested stakeholders. During the meeting, the product owner and the team will go through the vision, the roadmap, the release plan, and the *Product backlog* to develop a detailed plan for the upcoming sprint.

Together, the product owner and the team define a sprint goal, which is a short description of what should be completed at the end of the sprint. The success of the sprint will later be assessed during the sprint review meeting (See 2.2.4) against the sprint goal, rather than against each specific item selected from the product backlog.

Since the product owner decides the scope of a feature, the team usually requires the product owner to answer all of their questions to estimate the time it will take to finish a backlog item.

The team needs to be well prepared for this meeting. They should dedicate some time during a sprint to think ahead how the upcoming backlog items could be designed and implemented. Especially if the team is inexperienced and cannot produce a confident estimate of the effort required to complete a backlog item on their own and needs some additional information.

The team decides how much work they can successfully take into the sprint based on the team size, available hours, and the estimated level of

productivity. When the team has selected and committed to deliver a specific set of features from the highest priority items in the backlog, the Scrum master leads the team in a planning session to break these features down into *Tasks*, and form the *Sprint backlog*.

### 2.2.3 Daily Scrum Meeting

The daily Scrum meeting is a short meeting where the team members inform each other of what they have done, what they plan to do and if they have any impediments. This is to make sure that the whole team knows what is happening and to make sure noone is stuck alone with a single problem for several days. It is timeboxed to 15 minutes to ensure that the team spends more time developing than talking about developing.

### 2.2.4 Sprint Review Meeting

At the end of each sprint, a sprint review meeting is held. The first half of the meeting is set aside to demonstrate the potentially shippable prototype to the product owner that has been developed during the sprint. The product owner leads this part of the meeting and invites all interested stakeholders to attend. The product owner determines which items on the product backlog have been completed in the sprint. It is usually not allowed to use PowerPoint slides and spend more than two hours preparing for the demonstration.

### 2.2.5 Retrospective

After the sprint review meeting the Scrum team goes through a retrospective of the sprint with the Scrum master. The team assesses the way they worked together in the sprint and identifies things that went well and should be encouraged in the future, as well as things that could be improved. The results of this should be visible to the team during the following sprints. It is vital, therefore, that the feedback received at the retrospective be followed up on. Otherwise, the team may quickly see it as a waste of time.

## 2.3 How we did it

### 2.3.1 Scrum Roles

#### Product Owner

Jan-Henry Nyström, an Erlang Training and Consulting representative, was the product owner for this project, and acted as spokesman for Kimmo Gläborg, the *de facto* customer.

## **Scrum Master**

Several team members wanted to try being a Scrum master, so the role went to a different member at the beginning of most sprints.

## **Scrum Team**

There were no predefined roles in our team that we took on, but people tended to stick to the modules they developed and became 'experts' in their areas.

In order to ensure that we did not end up relying on something only one person in the team really understood, we ensured every task was reviewed by another team member, one that was not originally involved. We also tried to spread the competence using pair programming.

### **2.3.2 Sprint Planning Meeting**

Most of the time our sprint planning meetings went as planned, though some times the product owner was unavailable. In these cases the meeting simply needed to be scheduled one or two days later. These extra days would come in handy for cleaning up what we had produced the earlier sprint.

We used planning poker to reach a consensus on how complex each task would be. It is a "game" where the team votes on issues by placing a card upside-down, until everyone has placed a card - then they are all flipped at once. If the results are not unanimous, they are discussed, and usually require another round. Eventually everyone would end up voting on one of two adjacent values for each task, and we let the majority decide which we would go with.

### **2.3.3 Daily Scrum Meeting**

Our daily Scrums took place at 9.15. People could arrive as early as 8.00 and work until then, but as long as they did arrive before the meeting started it did not matter (formally).

We do recommend people trying Scrum to be really strict about the daily meeting since it is such a central part of the methodology. One possibility is to have the meeting at 8 sharp or 8.15, starting the day with the meeting. Then everyone would be able to start working directly, but you would miss out on the opportunity to recall what exactly you worked on the day before, as well as deciding which task to work on each day.

### **2.3.4 Sprint Review Meeting**

Our review meetings were always held on Fridays. The product owner would visit the team project room along with any other interested parties, and the

team would demonstrate new features on a live system, and answer any questions that might arise during the demo.

Usually we would spend one or two days before the demo checking if everything was working, and run test demonstrations internally.

### **2.3.5 Retrospective**

During the course we had several retrospectives. We wrote what we thought was good and what could be improved on post-its, and put all these on a whiteboard. Then every team member would put three dots next to one or more notes. When every member had done so, we counted the marks and tried to come up with solutions to the notes with most dots. A typical note could be “People commit code that does not compile”; we would then try to come up with a solution to that problem, e.g. “test if your code compiles before you commit”. It is important to make sure everyone knows that the team works as one unit so that no team-member feels that a note is all their fault.

## Chapter 3

# Milestones

### 3.1 Sprint 1

The first sprint we had no contact with the product owner and due to this we were somewhat in the dark. This sprint only lasted one and a half week, so we decided to do some research in areas we knew would come in handy, and some basic design.

### 3.2 Sprint 2

This was our first real sprint and the first time we had contact with our product owner. The duration of this sprint was three weeks.

We had many discussions about which implementation we should use in our cluster. We narrowed the options down to two possibilities, Inter-node communication or Map-Reduce. Since we knew that the timeframe for this project was 15 weeks we chose Map-Reduce for it's simplicity. To test everything we developed a simple wordcount program.

In the end of this sprint we had a working prototype which could solve Map-Reduce problems.

### 3.3 Sprint 3

The third sprint lasted three weeks, and was devoted to extending our prototype with functionality that updates the user on certain events, and finishing a raytracer program in OpenCL that our system could run. We also created a poster which we presented at the Erlang User Conference.

### 3.4 Erlang User Conference

Before the Erlang User Conference we spent three days entirely to further develop the poster and leaflets for our project. The conference went great and we got the opportunity to have rewarding conversations with many of the attendants, among them the creators of Disco <sup>1</sup>, an implementation of the Map-Reduce framework by Nokia.

### 3.5 Sprint 4

This two-week sprint was mostly devoted to adding the features of running background jobs in the system, and monitor the disk and memory usage across all connected nodes. Moreover, we started working on other user applications in OpenCL, such as image and audio filters.

### 3.6 Sprint 5

In the beginning of this sprint we decided that we could take more work than usual since all the other sprints had ended with very little to do. This backfired on us. We ended the sprint after two weeks with a cluster which could not run our existing example programs since we abstracted the I/O into a separate module. Apart from this we fixed the web interface which serves as the interface towards the users, and finalized some of the signal processing filters.

### 3.7 External Review

The external review was held in the end of the fifth sprint. This was done in the form of a oral presentation of the project done by two members of the team. Attending the review was experts in the field, both from the academic and industrial worlds, stakeholders and other invited guests. The presentation was followed by questions from the audience answered by members from the whole team.

### 3.8 Final Presentation

The final presentation was held on the 14th of januari 2010. This was done in the form of a oral presentation by one member of our team. This presentation was open for the public to attend. Following the presentation there was a short questions session and later a poster and demoing session outside the presentation area.

---

<sup>1</sup><http://discoproject.org>

## Chapter 4

# Resources

### 4.1 The Team



Figure 4.1: *The team - From the top left: Fredrik Andersson, Henrik Nordh, Niclas Axelsson, Vasiliij Savin, Christofer Ferm, Gustav Simonsson, Axel Andrén, Henrik Thalín, Björn Dahlman, Fabian Bergström*

The team consisted of ten computer science students, nine of which were native swedes and one an international student. We had different areas of expertise but noone had experience in a project of this scale which included

design, coding, integration, testing and documentation.

As a majority of the team were swedes, we used both English and Swedish when communicating within the group, though all important meetings were entirely in English.

## 4.2 Hardware

Every team member was supplied with a PC desktop workstation, which was mainly used for the development. All workstations ran Linux, most used the Ubuntu 9.04 distribution, while one person opted for running Gentoo. Apart from these computers we had an additional PC dedicated as a server for things such as revision control, issue tracking, common file storage and continuous integration. Our customer also provided us with four Mac Minis as the cluster hardware.

## 4.3 Training

### 4.3.1 Course in Erlang

In the beginning of the course we got the opportunity to learn Erlang from *Erlang Solutions*. Usually these courses are taught for two weeks, but due to the lack of time the teaching schedule was compressed to only one week. We learned a lot of Erlang and *OTP* during this training. Since most of the people did not know Erlang from before, this course proved vital for the project.

### 4.3.2 Course in Scrum

Apart from the Erlang introduction we also had a crash course in the methodology of Scrum, since we were planned to incorporate it in our development. The course lasted approximately a week, and was of immense help.

## 4.4 Tools

### 4.4.1 Communication

We set up a googlegroup to use as a mailing list for our team, though it was not used extensively. It was nice to have it when we wanted to contact everyone or if our product owner wanted to come in contact with everyone. We also collected contact information such as email and phone numbers which came in handy if people were absent.

The largest part of communication took place on our IRC channel. The benefits of using IRC were mostly the immediate response you would get, and the ease of pasting useful links for team members to visit.

#### 4.4.2 Version Control

Managing a project with ten developers without a **VCS** is practically impossible. We were instructed on how to use Subversion during the first weeks of training and did not think too much about changing to something else in the beginning. Although we did not have any great trouble with Subversion a distributed system such as GIT or Mercurial would have made it easier to move code from single developers without having to break the repository.

One member of the team used GIT to interface to the Subversion repository and that worked without any problem.

#### 4.4.3 Issue Tracking and Continuous Integration

For continuous integration we used Hudson. Hudson is rather basic but just having a continuous integration tool is powerful enough. In the early stages of the project we even had an IRC bot inside of the IRC channel which would alert us when the build was broken and list who was responsible. This led to a quick response from the responsible person which was great. Sadly the bot stopped working after a couple of weeks and we did not put any effort into fixing it which caused us to stop looking at Hudson.

The issue tracker we used for the project was Redmine. In the later sprints we also used the issue tracker to keep track of our Scrum tasks. We did so, because it allowed us to preserve the history of old sprints. This proved really valuable writing the final report, as we had all the important sprint information at hand. Also, post-it notes were too small to put all information related to the issue.

#### 4.4.4 Development environment

Most of us preferred using simple text editors, such as Emacs or Vim, though one person opted for using ErlIDE <sup>1</sup>.

##### Emacs

Emacs<sup>2</sup> is an editor with the excellent support for Erlang, both from official and third party channels. The team members using Emacs extended its functionality with some plugins:

---

<sup>1</sup><http://erlide.sourceforge.net/>

<sup>2</sup><http://www.gnu.org/software/emacs>

**FlyMake**<sup>3</sup> - Plugin for checking compiler errors and warnings before compiling manually.

**Erlware-mode**<sup>4</sup> - As an extension of the original erlang emacs mode which extends it with better skeletons that include edoc code and some fixes. Have many other useful features that we didn't use, we wanted it for its autogeneration of skeletons.

**Distel**<sup>5</sup> - "Distel extends Emacs Lisp with Erlang-style processes and message passing, and the Erlang distribution protocol. With this you can write Emacs Lisp processes and have them communicate with normal Erlang processes in real nodes."

#### 4.4.5 Code Testing

We started out using EUnit for our unit testing, and for that task it is a good tool. It has a nice a quite simple syntax and produces good error reports when a test fails. However when we started to look at integration testing EUnit was simply not powerful enough for our communication heavy testing. We then switched to OTP's CommonTest framework which was much simpler to write large tests in.

---

<sup>3</sup><http://flymake.sourceforge.net>

<sup>4</sup><http://code.google.com/p/erlware-mode/>

<sup>5</sup><http://fresh.homeunix.net/~luke/distel/>

## Chapter 5

# Problems

During the course of the project, we faced a number of obstacles, the most important of which are described in this chapter.

### 5.1 Not Being Proficient with Erlang

None of us had written a large program in Erlang before, and some of us had not written any Erlang at all. Even though the project did not grow very large, the complexity of having ten people working on the same code base was noticeable. If we had had more Erlang experience, we could probably have prepared better for a large project like this.

### 5.2 Nonexisting Backlog

One would expect to receive a complete backlog from the customer or product owner before starting a project of this size, but with our product owner absent, what we got was a list with a few guidelines. This meant we had to create a backlog of our own, using the very vague project description we were given. Thus we had to devote one and a half week to read up on several different subjects we had little to no knowledge of and do an overall system design before we could do any other work. This method obviously had some flaws, but we were also given a greater degree of freedom than normal, which certainly felt invigorating at times.

### 5.3 No Prior Experience with Clusters

Since we had no prior experience of implementing cluster frameworks, we did not know where to start. Should we write something with message passing like MPI? We eventually decided to use a map-reduce inspired framework, as map-reduce seemed easy to implement and we had a too limited timeframe for anything more intricate.

## 5.4 Task Division

We experienced problems throughout the project with having too little to do. This was related to how we subdivided our tasks in the beginning of the sprints, resulting in tasks that were poorly defined or simply too big to be solved immediately. These large tasks would still be worked on by team members towards the end of a sprint while others were left with empty hands.

## 5.5 Poor Testing and Continuous Integration Experience

We decided from the beginning to write tests for all our code. Unfortunately, no one in the team had any prior experience with any kind of testing framework. As a result of this we unwittingly wrote code that a unit testing framework could not easily test. Ultimately we managed to produce tests for a most of the codebase, but we could have been more efficient in our testing since a lot of time was spent rewriting tests that broke because of our coding.

We switched testing frameworks from EUnit to Common Test mid-project and a fair amount of time was spent redoing the tests to work with the new framework. Common Test was better suited to our modules, as they generally communicated much, which EUnit wasn't equipped to handle. but if we had been more up to speed with continuous integration, we could probably have saved time and caught more bugs earlier by using the right framework from the start.

## 5.6 Ignoring the Work Hours

A problem we experienced throughout the project was people arriving late. For a week or so we waited each day for everyone to arrive before we started the daily Scrum meeting, but eventually we decided to start the meeting even if not all team-members were present.

We also tried introducing a rule to punish latecomers, by forcing them to buy cookies or cake, which did get us some nice fikabröd but was not much of a punishment. So we tried having a wall of shame, where members would get ticks every time they arrived late. It may not be entirely fair for a person coming one minute late to get the same punishment as a person showing up two hours late, but we stuck to it. It didn't make much of a difference, though. Possible improvements could be to tick a person for each hour he or she comes late, and to wait with the ticks until the person arrives to make it extremely clear that they are receiving a punishment. We reset the wall every sprint though, to give people a chance to improve.

A related problem was team-members taking long lunches, in extreme cases an hour longer than intended. Also, we were supposed to work from eight to five, yet some members were coming in to work at nine and still ended their work days at five. However, in the long run, we never felt impeded by this as the productivity of the team was severely reduced in the late afternoons anyway.

## 5.7 Issues in the Working Environment

Our surroundings had some shortcomings when the project began. The ventilation in our room was half broken, and was not fixed until a few months later, forcing us to choose between breathable air or a comfortable temperature. We also had to contact the institution a number of times to fix issues with our key cards not letting some members of the team in.

As we were working with a cluster application, we also needed network equipment, which took several weeks for the IT admins could deliver.

## 5.8 Incommunicado

There were problems with our communication with the product owner. This can probably be blamed on both parties - partly because of him having other engagements, partly because we never really were in such a dire situation that we needed immediate answers from him. Besides the contact we had with our product owner, we also spoke with the customer himself from time to time, who was generally available. But it would occasionally result in some confusion regarding what to do, as they would request very different features from us.

Most of the time though, we made our own design and implementation choices. We mostly saw them on sprint review meetings.

Adding to our disappointment was the lack of any kick-off activities, which distanced us further from customer and product owner.

## 5.9 Scrum Master

Our method of choosing a new Scrum master every sprint did not give the Scrum masters a chance to reflect on his performance and improve, since most people only got one try, so deciding on one or two people to be scrum master from the start would have been better.

There were some issues initially, like forgetting to go through with specific parts of the methodology, but these all went away as we became more experienced in using Scrum.

Our Scrum masters did not spend that much time on Scrum activities. Their role was mostly to keeping our task board and issue tracker up-to-

date, writing down new tasks, and leading the Scrum meetings. Apart from that they were ordinary team members. We did not calculate the project velocity, as it was judged to be too much overhead for a project three months long.

## Chapter 6

# What we learned

### 6.1 Development Process

As we were working according to the methodology of SCRUM we had to define the length of our sprints. We experimented with different sprint lengths, ranging from one to three weeks. We found that three weeks felt most suitable for our team.

The first sprint was the shortest, approximately one and a half week, since we dedicated it solely to research and design. For this particular instance the sprint length worked fine, but during later stages of the project, it would have introduced too much overhead.

For the entire duration of the project we kept track of our progress using a task board. Later on we complemented this with an issue tracker on our server, and both of these were the scrum master's responsibility to update. We also used the *MoSCoW* system to some degree. According to the MoSCoW system, all tasks are assigned a priority [*Must be done, Should be done, Could be done and Would be nice* and people are supposed to work on higher-priority tasks before taking low-priority ones.

We performed pair programming occasionally and found it to be efficient, helpful, and reducing the amount of mistakes and bugs in the produced code.

One of our main ideologies was to use a continuous integration process during development. This partially worked, the problems were mostly due to our VCS not handling different branches in the repository well enough, so it was difficult to maintain a stable build all the time. We avoided creating branches, since we thought it will be difficult to merge them with the trunk later on. In the hindsight, we should have used a more advanced distributed VCS.

Whiteboards were used intensely for all design phases we went through. We mostly worked in pairs or groups of three when designing, and once a draft of a particular design was finished, it was presented to the whole team for feedback and input, until it became finalized. Overall, whiteboards are

extremely useful for brainstorming and designing.

We also let our server build the latest version automatically and continuously in order to make sure that everything worked as it should.

## **6.2 Project Rules**

Early on, we realized we needed some firm rules to keep the project going, so we set up some team rules. The most basic of these were the rules concerning our weekly schedule, with work days from eight to five and the daily scrum at 9.15. We also had a one-hour flextime possibility, thus changing the work hours from nine to six instead.

For keeping track of scheduled absence among the team members we used a dedicated part of a whiteboard where you could announce when you would be absent. This worked surprisingly well.

## Chapter 7

# Conclusion

The working environment and administrative tasks surrounding the project was both good and bad. The room was situated very nicely with a great view to the outside, but it was very cold due to the thermostat not working properly, so we had to choose between breathable air or a warm room.

We are quite pleased with the end product we produced. It mostly does what was planned from the beginning and throughout the different sprints, though later changes to the cluster were occasionally difficult to implement due to the original design being flawed.

The training before the project ensued was really good, however the pace of the training was perceived by some as somewhat high. The Scrum trainers did an excellent job during practice sessions to encourage group work, which contributed significantly to team building - at least for our team. We feel that more lecturing on different project tools such as Hudson would have been very helpful. Overall, the Erlang and OTP courses were good, and did not feel *that* rushed even though they were given in half the time they were supposed to.

Concerning the test frameworks we used we conclude that EUnit is not suitable for testing communication intensive applications, and we recommend using CommonTest for those kind of tests. We lost some time in the transition between these two, and want to emphasize the importance of choosing a good test framework from the start. Many modules in the cluster are also, by their very nature, hard to test as a single unit, as they depend strongly on other modules, if not the entire cluster, being up and running. This complicated testing, and it was in some cases quite hard to reach a full coverage of test cases for a module.

Our experience tells us that SVN was not fully satisfying and that GIT, for example, could have been suggested in the beginning of the course as a more suitable VCS.

In conclusion, aside from the experience of using Erlang, OpenCL and the many other tools used in the project, much was also learned about working

in a larger project. A greater understanding of group communication and group organization have likely been as great a benefit from the course as the technical knowledge gained.

# Bibliography

- [1] Henrik Kniberg, *Scrum and XP from the trenches*.  
<http://www.infoq.com/minibooks/scrum-xp-from-the-trenches>

## Appendix A

# Individual Contributions

**Henrik Thalín** In the beginning of the project, I did some research concerning math problems we could eventually implement, and worked with the initial protocol design. I was also interviewed by two groups of first-year students for their assignment concerning our project. When we started implementing our design I took care of implementing the database. However, after our first sprint of implementing, our cluster design had changed and settled quite a bit, and it was obvious the database needed a serious redesign. I was then involved in this redesign and implementation. For the remainder of the project, I have from time to time made revisions and additions to the database, such as adding the user handling. I also worked in extending the Statistician module to account for the disk and memory usage in the cluster. For the final weeks of the project I worked with implementation of some new user applications, namely the image and sound filters.

**Björn Dahlman** At the start of the project I did research on OpenCL, and worked on the design of the communication between Erlang and the external processes that would eventually be running. In the initial coding stages I mainly worked on the slave with modules for fetching and running tasks. Later on I did alot of “glueing” of the different components in the cluster when we first tried the whole cluster in it is wholeness. After that I worked on the Statistician and the IO Module and rewrote quite a bit on the slave. I have also done some small work here and there in modules like Listener, Dispatcher, Db and implemented smaller things like NetMonitor. During the work with the IO Module I did alot of work on getting Riak up and running and making it compliant with our cluster.

**Christofer Ferm** In the begining of the project I worked with researching possible hardware to use for the project. I investigated and compared the Mac Mini's against Nvidia ION based platforms. After that I did some research on OpenCL and I wrote the first simple example program using OpenCL, it just added two arrays and printed out the result. After that I started to work on the raytracer. It took some time to get the hang of OpenCL so this task has been with me throughout out the project. I have also worked on adding functionality for background jobs in the cluster. I later researched Image filters and worked on the Image filtering application. I also researched Audio filtering and worked on a simple program wich adds echos to a wave file. Then I started porting the image filter application to use OpenCL. I was Scrum master for our fifth sprint. My main area throughout the project has been working on cluster applications using OpenCL.

**Gustav Simonsson** In the early stages of the project I did research into OpenCL and Erlang. I worked on the overall design of the modules of the cluster, in the stage before we began implementing them. After this I worked on the ECG module until it was later redesigned. I designed and imlemented the statistician module; later on I worked on its alarm functionality. I also worked on the master supervisor. Later on I researched a few different backends for parallel file-systems and added/clarified comments in a bunch of other modules. Finally some minor refactoring of various parts of the code and validation of tasks done by other project members.

**Fredrik Andersson** I started out doing some hardware research. We investigated the Mac Minis and a couple of Nvidia ION based platforms as potential candidates for the project. When implementation began I worked alot on the slave node making the initial application and writing the first basic supervisors for that. During sprint 2 I was interviewed by the first year students on the computer science program. I also investigated on how to create OTP applications since we were all new to it. During sprint three I took the role as scrum master. My main responsability was the logging system Chronicler that we developed, it is probably the single part of the system I have spent most time on. The first solution was put togheter quite fast but it proved not to be supporting the stuff functionality wanted in the web interface so I had to rewrite larger parts of it near the end of the project. I also worked on the web interface where I did the log viewing page and the filtering mecanishm found therein. In the end I also held the final presentation.

**Axel Andrén** Early on while we were still researching and unsure of what to do, I made the heartbeat module, a module for keeping track of which nodes in the cluster are alive. But we discarded it when found that there was a feature like that built into Erlang already. I focused on the dispatcher instead, but it was given over to other team-members before the sprint was finished.

The first major thing I worked on was the Statistician module, which I have been helping maintain throughout the project. It's seen a lot of additions and updates as we thought of more things the user might want to know.

Then there was less coding and more getting PVFS to work, which was a real pain - partly because of a bug that I found that prevented us from running it properly. But it was fixed soon enough after it was discovered, and once we got past the lengthy setup on every computer it works well.

On the side I have been fiddling with EDoc and other documentation, trying to keep it up to date as well as not crashing our compilation process. There was also a number of tests that were causing issues, and so had to be rewritten - now they are almost all Common Test tests, rather than EUnit kind of tests, which were not appropriate for this kind of software.

**Henrik Nordh** In the beginning of the project, like everyone else, I was doing research. My area was other clusters and distributed systems. During this period I looked a lot into the DISCO project by Nokia. After that the design phase of the project began. During this period I was involved in the Master node design. We also thought long and hard about the Erlang-OpenCL communications.

A redesign of the database was done in the third sprint. I was involved in that redesign and implementation of the new version. Later I have made various additions to the database module. I also started to work with the tests, most of the test coding was done with the help of pairprogramming. After that I added support for background jobs on the master node and the DB module. During the length of the project I have been doing validation of various tasks and testing.

I was involved in the poster for the Erlang user conference and I made preparations for the trip to EUC in Stockholm. I was responsible for, and held the presentation on the external review.

I was appointed "security officer" for the entire course, this involved trying to get everyone access to the rooms and building and talking to the administer of the card system a lot.

I also acted as the group leader for our team, this involved various

administrative tasks such as holding a presentation for the first year students about this course, and also getting interviewed by these same students about the course and the projects.

**Vasilij Savin** I was the first Scrum master for our team during the first two sprints. It was not the best idea, since we struggled a bit getting Scrum running well. In the beginning, I was also working on the general system architecture and master node design. I tried to understand different components needed for our cluster and how they would interact. Also during the first sprint I studied principles of distributed systems design.

During the development phase I worked mostly on implementing the master node components. My major responsibility were the Dispatcher, the ECG and later the Listener modules. Eventually, I also converted the master node into a OTP application. As everyone in the team, I was also writing tests for the code I was developing. That proved to be a difficult task, since I really struggled with the EUnit tests, but once we switched to CommonTest it became a much more pleasant experience. After the second sprint, I was involved in database redesign during design phase. During the final sprint, I was mostly fiddling with the slave node code, refactoring I/O and streamlining the code base. I have implemented the abstract I/O functionality and facilities to develop the new storage plugins, as well as developing reference implementations for NFS and Riak based storage solutions.

I was also involved preparing our marketing materials - the poster, presentations, and the leaflets. In addition, I delivered the project presentation during the external review.

**Fabian Bergström** I have been involved in several places but not really had a clear main area of work. I did some research of solutions for distributed computing in the beginning, then I worked mostly with the general design of the system, and on the design of our map-reduce implementation.

Later on I implemented additions to the server for handling background jobs. I also worked on the design and implementation of the communication between our cluster and the applications, both in the previous file system based incarnation and later when we moved to communicating on the standard input and output streams. I wrote our ruby scripts to glue the cluster together with the OpenCL code.

I was also the Scrum master for our fourth sprint. The sprint was only two weeks, so it was quite stressful at the end. We had no significant impediments for me to handle during the sprint, so I was mostly

synchronizing our task board with our issue tracker and updating the burndown.

**Niclas Axelsson** In the beginning I was involved in many of the initial discussions of the cluster design. Later on I researched on how to measure power consumptions on our mac minis and designed a module that estimated it. I have also worked on the listener and web interface. In the end I was involved in installing PVFS. I also had the role as system administrator.