Course report: Trapexit 2.0

Yury Dorofeev, Jacob Ericsson, Hariprasad Hari, Jonas Rosling,
Niclas Stensbäck, Samuel Strand, Wilson Tuladhar, Yeli Zhu

February 28, 2011

Contents

1 Introduction **5**

2 Methodology **6**
2.1 Scrum . . . 6
2.1.1 Scrum Roles . . . 6
2.1.2 Sprints . . . 6
2.2 Scrum, our way . . . 8
2.3 Team building . . . 8
2.3.1 Team rules . . . 8
2.3.2 Kick-offs . . . 9
2.3.3 Schools . . . 9
2.3.4 Daily stand-up . . . 9
2.3.5 Spokesperson . . . 9
2.3.6 Fika . . . 9
2.3.7 Games . . . 9
2.3.8 Movies . . . 10

3 Timeline **11**
3.1 Sprint 1 . . . 11
3.2 Sprint 2 . . . 11
3.3 Sprint 3 . . . 12
3.4 Sprint 4 . . . 12
3.5 Sprint 5 . . . 13
3.6 Sprint 6 . . . 13
3.7 Sprint 7 . . . 13

4 Resources **14**
4.1 Team . . . 14
4.2 Hardware . . . 14
4.3 Software . . . 15
4.3.1 Redmine . . . 15
4.3.2 Git . . . 15
4.3.3 Hudson . . . 15
4.3.4 Rebar . . . 16
4.3.5 Testing . . . 16
4.4 Local Amenities . . . 16

5 Problems and Issues **17**
5.1 Bottlenecks . . . 17
5.2 Scrum Master . . . 17
5.3 Product Owners . . . 17
5.4 Lacking documentation . . . 17
5.5 Workflow . . . 18

6 Conclusion **19**
6.1 What we have learned . . . 19

A Individual Contributions **20**
A.1 Hariprasad Hari 20
A.2 Yury Dorofeev 20
A.3 Jacob Ericsson 21
A.4 Jonas Rosling 21
A.5 Yeli Zhu 21
A.6 Samuel Strand 22
A.7 Niclas Stensbäck 22
A.8 Wilson Tuladhar 23

List of Figures

1 The Scrum process 6
2 Our Scrumboard 7
3 Our team 14

Abstract

This document describes the methodology and development process of the project which resulted in new version of the community web site trapexit.org. Project methodology, team resources and a timeline is examined as we present one semester of full time work for eight computer science students.

1 Introduction

Like most programming languages, Erlang [1] has many community websites, one of them being Trapexit.org [2]. It is currently managed by Erlang Solutions [3], and this project deals with laying the groundwork for an overhaul of that website.

Making a website entirely using Erlang [1] can be quite easy or annoying, depending on how you approach doing it. Web frameworks such as Erlang Web [4], Nitrogen [5], Zotonic [6] and others make it easy to host your own webpage based on Erlang [1] if you wish it. What we do, though, is something quite different.

Our main goal, throughout the project, was to make a system backend that is robust, modular and (where possible) fast. Our main result is having constructed a website that relies on a message bus and through many layers of abstraction still manages to handle large amounts of traffic efficiently.

2 Methodology

This section explains the Project Methodology Scrum and how Scrum worked for us.

2.1 Scrum

This project utilizes Scrum [18]. Scrum is a agile method for development, used primarily in software development that drew on inspiration from the "New Product Development"[7] and is the work of Jeff Sutherland and Ken Schwaber. They first presented Scrum at the OOPSLA [8] of 1995. Since then, they have worked on gathering and writing down the best practices into what is now known as Scrum.

Time in Scrum is divided up into so called sprints (see figure 1) and there are several roles and rituals required of a team using Scrum. We will detail them below.

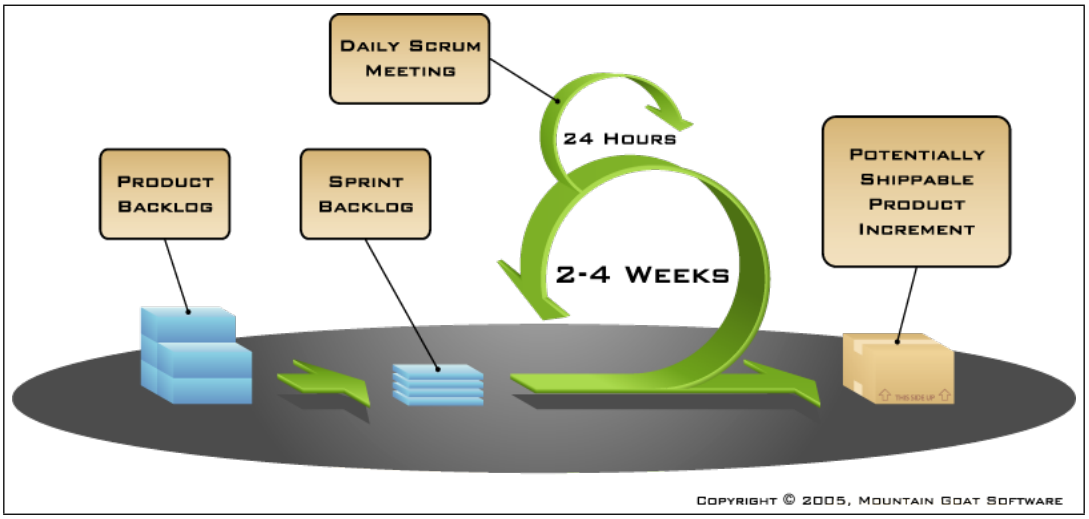

Figure 1: The Scrum process

2.1.1 Scrum Roles

- **The Team** is the one responsible for implementing the wishes of the product owner. The recommended team size is 5-9. Should teams grow larger than that, it is recommended to divide them into subteams.
- **The Scrum master** is an outsider to the team that enforces the law of Scrum onto the team. He is the one that makes sure all the meetings adhere to their timeboxes, that the routines of the daily stand-ups follow the format and so forth. Basically, the Scrum masters role is to protect the team from distractions and make sure they use the correct methodology.
- **The Product owner** is the customer purchasing the services of the team. He is the one dictating the bounds of the project by writing user stories (short explanation of what should be done during the project, some details toward particular components and system requirements) and places them into what is known as the backlog.

2.1.2 Sprints

A sprint is a period of time typically ranging between 2-4 weeks. This is the main time unit in Scrum, nearly everything you do revolves around your current sprint.

At the beginning of sprints there is sprint planning. This is arguably the most important day of the sprint. During the sprint planning day, the team together with the product owner agree on the amount of work that can be done during the coming sprint, define it properly as to avoid misunderstandings between the product owner and the team. After the objective is set up and the team agrees that they can deliver the product owners request at the end of the sprint, the team starts dividing up the user stories into tasks.

A task in Scrum is a piece of work that should not take one person more than a day to do. Typically the team grabs a couple of post-it pads and start penning down what seems like reasonable tasks to everyone. In the next stage the team time estimates all the tasks, doing their best to assign the expected number of man hours required to complete the task, for each task. Usually, this is where a lot of disagreement and the following the discussion takes place when members of the team do not agree on how hard a task is to do.

This is typically indicative that the task is not properly understood by all team members and needs to be clarified. The product owners also sit in on the time estimation, but only speaks up if he feels that the team has misunderstood a task, a user story or a something other about the project as a whole.

Lastly comes prioritization, when the teams decides the relative importance of all the tasks and sets up the tasks on the Scrum board (A whiteboard or a bulletin board where you keep the tasks. See figure 2). When this is done, the sprint planning is complete.

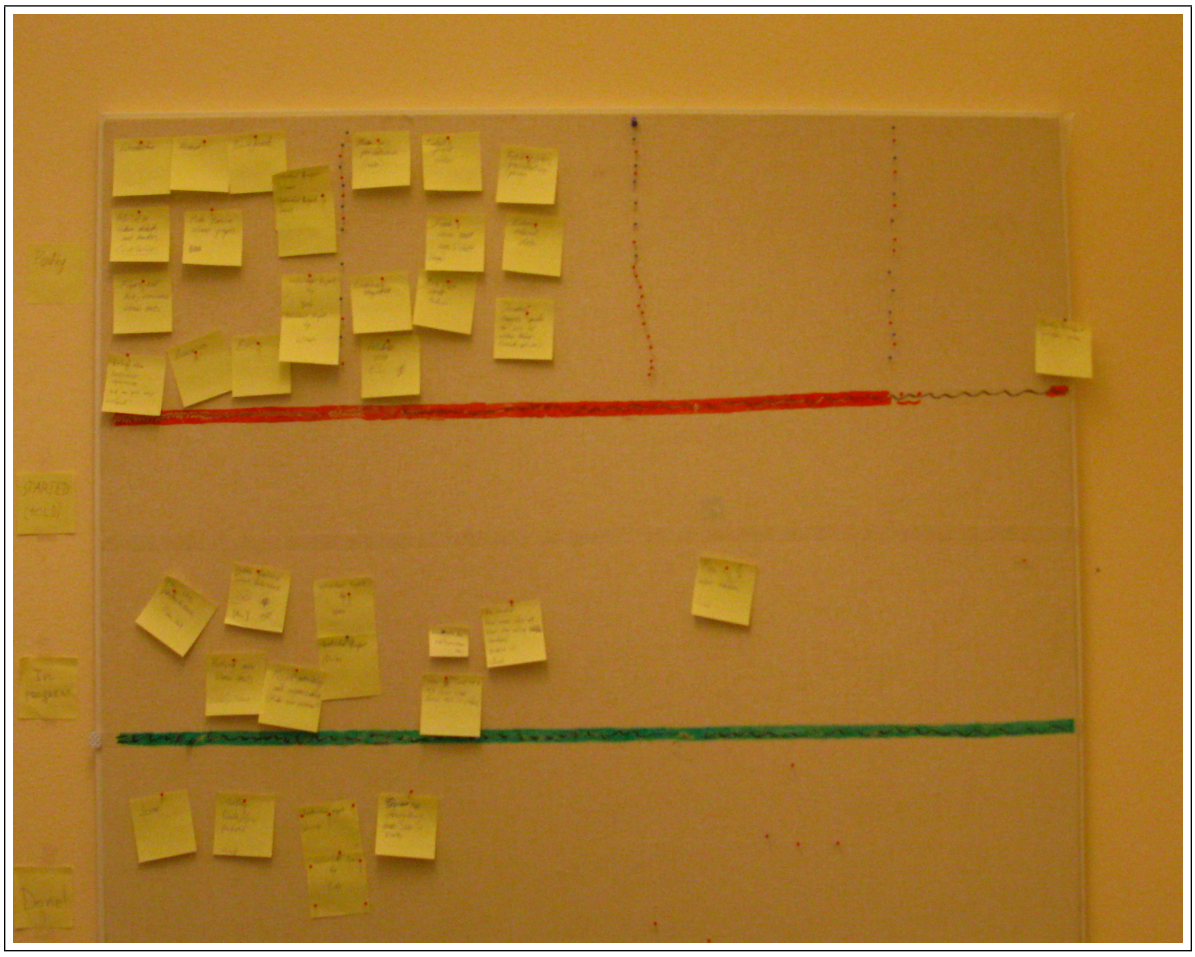

Figure 2: Priority is descending from the left to the right and the height denote the state of completion where above the red(top horisontal) line is pending/unstarted and below the green(bottom horisontal) line is finished

Then the daily work proceeds during the rest of the days of the sprint and at the end of the

allotted time, the product owner will come visit and expects a demonstration of what the team has achieved during the sprint. This is done on the last day of the sprint.

After that, the team wraps up doing a retrospective of the sprint, which is a meeting where all the team members get to talk freely about what did and didn't work well during the sprint and how to fix that until the next sprint.

2.2 Scrum, our way

We did not change much from the Scrum template in our implementation of it.

Perhaps most notably, while we did time estimation with planning poker (A simple system of time estimation. It is a deck of cards where instead of normal playing cards the abstract numbers are used. The numbers are distributed from 0 to infinity. 0 means that the task is trivial and might be done immediately) initially, we found the entire process to be tedious, time consuming and ultimately irrelevant to our ability to perform during the sprint. As such, we replaced it with the following system of "time estimation".

After task generation, we looked at each of our tasks and asked ourselves a simple question, "Will this take more than a day to do?". If the answer was yes, then we broke it up into subtasks, otherwise we left it alone. This also meant that we did not have a Burndown chart (graphical representation of work left to do. The outstanding work is on the vertical axis and time on the horizontal. The chart starts at the left up corner and ideally should end up in the right down one), on the other hand we could never use one before either. The reason for that was that the Burndown chart add-on to Redmine(see section 4.3.1) never worked for us and we found it too time consuming to do it manually.

Sometimes, a discussion broke out whether or not a specific task *could* be done in a day or not. In the end, almost all of our decisions were consensus decisions after a brief discussion. This method achieved the goal of the entire team agreeing which tasks are "big tasks" and which are not while you also get to split up the big tasks into smaller tasks, all the while saving a lot of time. If we had had an outside Scrum master, this probably had not been allowed. We were pretty happy with the arrangement, but maybe we were doing it wrong in the first place. One of our project owners, Henrik Thalin, who took this course the previous year said that they were doing sprint planning meetings in four hours tops. We could easily go on for eight hours and still having to race at the end of the day to be finished before our timebox was up.

Also, since we did not have a Scrum master external to the project, we ended up doing all the work that the Scrum master should have done (putting tasks into the Redmine(see section 4.3.1) and administrating the Scrum process).

2.3 Team building

One of the important lessons that we learnt through Scrum is working in a collaborative environment.

2.3.1 Team rules

We followed some strict rules in our project.

- Work hours is 8.00 to 17.00
- Weekly flex, to be balanced on Friday when you leave.
- Daily stand-up at 9 am
- Late to stand up - bring Fika
- Communication - English ONLY

- Ill - Inform the Spokesperson

2.3.2 Kick-offs

At the start of our project, the University arranged a pool night for us. This was our first informal interaction with our product owners. This event was common to both project teams. Another kick-off was arranged by our company. We went bowling at Bowlaget. These made us to get closer and friendlier with our project owners.

2.3.3 Schools

It would be difficult for everyone to make a research on these things and get acquainted with these tools. So we had decided that a single member of our project would go for a single tool. That person would make all the necessary research. We then prepared a time-slot in our dashboard to have that school. During the school the rest of the team-members were familiarized with that tool. It was also on occasion attended by our teaching-staff. Schools were considered an efficient way to learn more and we held schools for many tools.

2.3.4 Daily stand-up

According to the Scrum methodology, we devoted at most 15 minutes every working day to discuss about what we did the previous day and what we are going to do the current day. It is strongly recommended to do these meetings standing up in order to speed up the discussions and make them more effective. During the meeting each team member had an individual speech. The technical details were not discussed during the meeting. If a team member had any kind of problem or was stuck somewhere and needed help, it was to be emphasized during the meeting.

2.3.5 Spokesperson

At the beginning of the project we chose one team member who was responsible for communication with project owner and supervisor. When we had problems or questions we redirected them to our spokesperson. This person had to be responsible, self motivated and representative. Fortunately our spokesperson had all these properties.

2.3.6 Fika

According to the old Swedish traditions coffee brakes with something sweet and taste improve the working process and social atmosphere dramatically. Fika has been our major highlight for our team. Everyone also posted their recipes in a separate wiki that we called “Project Recipous”. Normally we would have weekly Fika on Thursdays, usually at 15.00. We fixed on Thursdays because we have our demo day on Fridays, so it would be nice if we all were energetic on the afternoon before the demo day. Apart from the regular Fikas, we also had late Fikas. These are given by those who were late for the daily standup. Late Fikas worked out very well for us, as all our group mates were there during the first four sprints. After that, there were some issues concerning the winter weather. We would normally have our Fika break for about maximum of 30 minutes.

2.3.7 Games

It is difficult to just sit and work in front of computers for eight hours throughout the week. It would really make us insane. Some physical activities are always necessary to keep our minds refresh and energetic. Whenever we were at high pressure due to coding or due to some major problems which could not be solved or due to time constraints, we would play table-tennis. It

just suited us and all enjoyed at the most. During the lunch hours we also played board games like “Arkham-horror” which made to think and go made due to its complexity. We also had a gaming night at one of our friends house and it was fun. The game called “Mafia” was truly classic and excellent. These games made us to be more enthusiastic in our work.

2.3.8 Movies

Apart from playing games during lunch hours we were also watching movies. It was again a different experience for us to know about the culture and the media background of different countries. Since the movies are long, it took us two to three days to finish watching it fully.

3 Timeline

In this chapter we will show the general timeline of our project, with details on the acitivities undertaken on all sprints. Each sprint lasted roughly two weeks.

3.1 Sprint 1

The first sprint was devoted to discussions and researching of relevant components, their pro's and con's. We had close contact with our product owners. They gave us user stories about the product they wanted to get and advices regarding Scrum methodology 2.1 based on last year's experience. Moreover, we managed to use a message bus which would be usable for sending messages between nodes.

Scrum master: Hariprasad Hari

Sprint Goals:

- Find relevant components.
- Find out how they work.
- Design the architecture.

Demo goals:

- Oral presentation about the tools that we had chosen.

3.2 Sprint 2

During the second sprint we decided which database to choose, discussed our public API design and extended the message bus functionality. After that the general message structure was defined and test methodologies were explored. We had a large discussion regarding our architecture. We came up with an architecture having double abstraction as the key feature.

Scrum master: Jacob Ericsson

Sprint Goals:

- Web Framework - Decide upon API and structure (division of Labor) and start implementing.
- Database - Finalize API & regards with Web Framework(implement).
- Message Queue - Finalize API & regards with Web Framework (implement).
- Authentication Module - Create module and API. Decide on Database for authentication module.
- Discuss practicalities like coding standards.

Demo goals:

- Show off the good architecture.
- Show off sending and receiving of messages between various components using Message Bus.

3.3 Sprint 3

Third sprint was devoted to show the working build server with 40% test coverage reports, a working Planet Trapexit page. We finalized files placement structure and defined the web framework protocol. Much more coding was done on that sprint as we developed each of our API's for our modules. Moreover, we had some practicalities such as Rebar(see section 4.3.4) school.

Scrum master: Samuel Strand

Sprint Goals:

- Get a build server. Machines that run automated tests.
- Finalize Database API. Decide which available API to use to access the Database.
- Make a site Service "Planet Erlang".
- Research HTML Templating.
- Look into CMS(Content Management System) & Web Crawler
- Complete Authentication Module.

Demo goals:

- First working version of Planet Trapexit website. Login, register, edit info should work and put placeholders in place of other functionalities.
- Show a working build server, with test coverage reports

3.4 Sprint 4

It was the main and most productive sprint in our project. We made a poster for the conference and trained each other by asking questions regarding our system. Planet Trapexit website was polished. We were well prepared and represented our project successfully. During that historical day we had exciting conversations with developers, saw legends of Erlang community and had an unforgettable dinner. After the conference, start-up scripts for our components were written.

Scrum master: Niclas Stensbäck and Wilson Tuladhar

Sprint Goals:

- Prepare for the Erlang User Conference. Make a poster, prepare to answer questions, set up a laptop to show the demo.
- Finish up the planet, should be 95% finished featuring nice web design, RSS feed aggregator [19] and possibly other services.
- Implement a templating standard.
- Make a Forum/Wiki service and integrate it into the system.
- Work with testing. The goal is 70% test coverage.

Demo goals:

- Show off our work at the conference!
- Show off test coverage reports from Hudson(see section 4.3.3) at the conference.

3.5 Sprint 5

Sprint number five was devoted to stress testing and improving the system stability. Stress testing was done with a tool called Tsung(see section 4.3.5) while stability was reached by implementing supervisor behavior. According to the product owner requirements, the code should be properly commented. Thus, one part of that sprint was spent on writing Edoc [20]. Another sprint goal was to evaluate given Content Management System [21] and incorporated it into our system.

Scrum master: Jonas Rosling and Yeli Zhu

Sprint Goals:

- Integrate a CMS(Content Management System) [21] into the system
- Stress test the system
- Improve system stability

Demo goals:

- Show off CMS(Content Management System)
- Show fancy graphs of stress tests results
- Show before/after results of system stability improvements / refactoring.

3.6 Sprint 6

During this sprint we finished with writing Eunit [14] tests and reached 85% test coverage for all our modules. It helped us find and investigate some bugs. Then, parts of the source code was re-factored. Moreover, a project review was held. Two members of each team presented the teams work. The review was attended by experts from the academic and industrial fields.

Scrum master: Yury Dorofeev

Sprint Goals:

- Prepare course presentation
- Get everything ship-shape. Have a 70% test coverage, do some bug fixing, prepare the documentation
- Start on the Report.

Demo goals:

- Show off test coverage report.
- Show off module documentation.
- Make readme.txt files for all the scripts.
- Show off report (if started).

3.7 Sprint 7

We spent the last sprint on writing product and course reports. The final presentation was also held on January 13, 2011.

4 Resources

This section details the Resources that helped us in developing our system.

4.1 Team

Figure 3: Our team - From the left: Wilson Tuladhar, Yeli Zhu, Jonas Rosling, Jacob Ericsson, Niclas Stensbäck, Samuel Strand, Hariprasad Hari, Yury Dorofeev

Our team consisted of eight computer science students. This year the number of native Swedes students decreased dramatically compared to the previous year. There were four Swedes, one Nepalese, one Chinese, one Indian and one Russian student. We had different backgrounds, we were in different age but none of us were involved in such kind of project before. In our group we used English for both important project meetings and non project communications.

4.2 Hardware

Each member of the team was supplied with a desktop PC of 2.66GHz Intel Q9400 computers running with 3GB RAM built by HP loaned to us from Uppsala University. We also had four extra machines for integration & testing with Dual core processors. We used one of those severs for code integration and version control (GIT 4.3.2) and also for Redmine 4.3.1 access for documentation, one for the Hudson 4.3.3 for automatic build and test of our system and rest for the testing purpose. All the machines were running under Ubuntu 10.x with updates to the OS which have been applied as they have become available.

4.3 Software

During the course of our project we have used a lot of different software that were not included in the product directly. Some of the tools that we used for developing the product are as follows:

4.3.1 Redmine

Redmine [10] is a project administration tool, sporting a wiki, a digital Scrum board, the ability to view the Git repository [9] and other plug-ins useful for managing projects. We installed Redmine [10] to one of our server machines and got access to the team as well as the project owners. Initially we were only using the Wiki in our Redmine [10]. It proved to be very useful as we were able to update our stuff as soon as we did something. Our research on various tools, major discussion, system architectures, conclusions on various problems, running and configuring the system etc were all posted on the Redmine [10] wiki. This also helped us in writing our reports. Some plug-ins like the forums were not used as we thought it will be needless to use forums to communicate when all our members are sitting in close proximity.

We also added all of the user stories and the sprints' tasks into the Redmine [10]. This was very useful as we could manage the sprint in a simple manner. In this way we can keep track of the states of the task and accordingly we can focus, keeping the demo in mind. You can also see which task is done by whom, and to what percentage the task has been completed. You can also log the spent time, i.e the time taken for the tasks. Another advantage is that we can view the history of the project.

We made our Redmine [10] server accessible to our product owners so that they could checkout the team progress without having to come to the university to check up on us.

4.3.2 Git

We used Git [9] for our version control of the system. We used one of our server machines for Git [9] as mentioned in the Hardware section 4.2. Only the team was given direct access to the Git repository [9], even though our project owners could view the code via our Redmine [10].

It has been very useful in our project, as each one can clone the repository locally to their systems. The members can do changes locally which doesn't affect the global stable repository unless otherwise he/she uploads the changes to the Git [9] server. The good thing about Git [9] is that we can create our own branches, work on it and then rebase to the stable branch. By this we can avoid conflicts to our remote stable branch.

We had quite a few troublesome times when the Git repository [9] would be broken or someone's local branches would be corrupted. Though Samuel Strand was generally able to resolve these issues, it would take time and energy from people since they couldn't work as they wanted. It certainly saved us time in the end though.

4.3.3 Hudson

Hudson [11] is a continuous integration tool which we used for our integration testing. Initially it required some research, but it was easy to get up and running on our systems. After working well in the local machine, we installed Hudson [11] in one of the server machines. We configured it so that it takes the source files from our Git repository [9]. It compiles them using our building tool Rebar(see section 4.3.4). We have configured Hudson [11] to build our repository every 5 seconds. After every build, we can check for any compiler warnings if it was there or otherwise it would be a successful build.

We never really used Hudson [11] as much as we would have liked since it had no way of interacting with the team members. There were sound plug-ins and such but none that we could find worked. This made the Hudson [11] in essence a waste of time since it could not make the build status known to the world.

4.3.4 Rebar

Rebar [12] is an Erlang [1] build tool which we used for easy compilation, testing of our applications, generating the documentation through Edoc [13] and handling releases. Rebar [12] uses standard Erlang/OTP conventions for project structure.

Rebar [12] provided support for most of our development such as:

- Compilation,
- Eunit [14] and providing it's coverage analysis, and
- Document generation through Edoc [13].

4.3.5 Testing

We decided upon doing unit tests for the whole system and we chose EUnit [14] from the Erlang [1] libraries for this purpose. EUnit testing [14] produces nice error reports specifying the place and cause of the errors. But since our system is a concurrent system with a lot of side effects, we needed to spoof functions and modules beside the one being tested in almost every unit test. For this we used Meck [15], an Erlang [1] module which is used for spoofing the methods. And this worked out quite nicely for us. For the next level of testing, we had to test how much load our system could handle i.e. load testing. For this purpose, we chose another Erlang [1] built tool called Tsung [16]. Tsung [16] is a tool built to test the scalability and performance of the client/server applications. It is used for stress testing the servers. To test our website, we sent some requests for the pages to the server at certain interval of time which were specified in the Tsung [16] configuration file and when we ran Tsung [16] it ran all the cases in sequence and gave us the result in nice graphical design and we could see how different components of our system behaved.

4.4 Local Amenities

Apart from the hardware resources, we had been provided with the following:

- Separate Project room.
- White boards.
- Bulletin boards.
- Stationery items as Duster, Pen, etc
- Recreational items as Coffee, Tea-bag, Sugar, etc
- Acoustic walls
- Printer (shared between both project teams)

It was very useful for us to do our project in a calm environment. Only the members of the project, the responsible staffs and teaching assistants had access to our project room. We used our bulletin board to be our Scrum board and the white boards for a variety of other purposes.

5 Problems and Issues

5.1 Bottlenecks

During our project at the end of most of the sprints we had a problem with bottlenecking. The problem was that we found that we only had two or three people actually working on anything. This was because people were waiting for results from each other when the end sprint was approaching. We eventually got better at not bottlenecking. One thing we did was to be stricter when breaking down the user stories into tasks, making them smaller and thereby making more tasks. Another thing was that we started to prioritize the tasks differently, making it easier to choose tasks in a way to avoid bottlenecking. We felt that the bottlenecking was also due to the fact that some goals were very vague. People tended to avoid these tasks since it felt "scary" to pick them up and start working on them.

5.2 Scrum Master

The problem with the Scrum Master is that we had our own internal Scrum Master. According to the Scrum methodology which was explained in chapter 2.1 we should have a member who is outside of our project and not one of our project members. By having an internal Scrum Master, he/she can be biased and would be unfair in some cases. There are chances that the Scrum Master itself may not abide by the rules framed by a project. Fortunately this was not in our case although we had an internal Scrum Master.

5.3 Product Owners

Speaking about Product Owners as a problem would be somewhat awkward or not so good. But keeping in mind that this report would be for future students we are writing on what we thought regarding our product owners.

We had a nice relationship with our product owners in the beginning of the project. As we mentioned in Kick-off chapter 2.3.2, they were good for four sprints. After the Erlang User Conference, things didn't work out well. The communication between us was lagging as we were shocked that they didn't turn up for the next two sprint demos.

Sprint planning for sprint five did almost not happen due to the product owner being late by several hours and neither telling us about it nor responding on the cell phone. For the sprint planning meeting for sprint six we were without product owners entirely. This not only made our work a lot harder but it was also a pretty bad blow to our working morale.

It's a not a sign of respect that one of the product owners stopped responding to our emails and none of the others were covering for him. But the worst part was when none of our product owners were present for the technical review on December 17. Later we found out that one of them was ill. Apparently he did not inform the company about this so they were also unable to know that he needed cover.

5.4 Lacking documentation

Sometimes when you work with open source software, you will run into the problem that everything is not quite as well documented as you would like. Many developers feel that there is no need for an extensive documentation and would rather direct users to either mailing lists(the slow but functional alternative) or the source code itself(the painfully slow alternative). While you will eventually get your software to work this way, it will take up a lot of your time and both these options are readily eclipsed by just having a documentation.

For example, CouchDB [17] has an extensive wiki as well as a free online book of over 200 pages that users can partake of should they require information on some specific details in the workings of the database. Reading a book or a wiki is faster than reading source code, and in

a time-boxed project that will not have more than four months of development, time is of the essence.

On the other hand, Rebar's (see section 4.3.4) "documentation" is limited to a short Read Me and an example config file, and figuring out how it worked required time consuming trial and error. Had there been a more well documented option for rebar, we most likely would've saved time using it.

5.5 Workflow

One of our major issues in the project was the workflow. In the beginning we were all working on the same branch with different issues resulting in nobody knowing if it was their code that was broken or if the system did not work. We were sort of late in dealing with the problem and did not come up with a solution for it until around sprint 5 when we actually started making separate branches for every issue that needed to be solved and when the issue was solved then only merging it into the main branch.

6 Conclusion

6.1 What we have learned

For all of us Scrum [18] was a new and unfamiliar methodology. During the project we learnt lots of interesting and useful things such as:

- how an IT project evolve in a small group of developers
- how to organize working environment in proper way - we placed our tables in such a way not to disturb each other and had easy access to boards
- how to improve efficiency of working process - at the end of each sprint we made sprint retrospective where we discussed what were done well, what were done bad and what should be done to improve our weak points
- how to work without manager and every day inspection - group of students were placed to an isolated room with coffee and sugar instead of teacher and demo days instead of exams
- why Scrum actually works - at the first sprint we felt quite pessimistic regarding the project success. The project scale and complexity seemed to be enormous. Then we split the whole project to short periods and assigned a fixed number of goals for each period. After that we divided each goal to the number of small tasks and started to solve the problems. Optimistic moods grew up
- all the stages of development - we saw the development process from the beginning to the final working version
- testing - we saw in practice the importance and complexity of testing
- research and analysis - plays extremely important role in the development. It is better to spend long time on research than reinvent the wheel
- how to communicate within multi-national society - to create something awesome, stable and effective in a group of people from different cultures and with different mentalities, sounds unrealistic but it works
- cooking traditions and features - Friday's fika is the best way to get to know national kitchen

A Individual Contributions

A.1 Hariprasad Hari

It was a nice experience for me to start our project with me being the Scrum master for the first sprint. Although we had a Scrum course from Klarna before the start of our project, it took some time for me to adapt to the scrum methodological aspects such as the daily standup, sprint planning etc. It was good for me, as it developed my skill to work in a collaborative environment together with my project mates.

During the first sprint, as similar for every project team, we also started to do some research work for our components, tools etc. My major part was mostly in databases (along with Yury), supervisors and application models. For the research, I read about databases like CouchDb, Riak and content management system like Drupal, Zotonic. Concerning the databases we need to figure out the best fit for our architecture. I and Yury designed the database schema architecture. It explained how the schema for the forums should be. In the future sprints, we started implementing the API's for the database(internal database api). I also wrote the database connector which was latter polished by Jacob and Jonas to be a gen_connector with the db_callback module. Initially I also worked along with Samuel, in deciding the web framework protocol. I did the Redmine school along with Yury.

After that I switched to make the system to be fault-tolerant by implementing supervisors and making our components to be individual applications. For the supervisor, I proposed the supervision tree but unfortunately couldn't implement that model as it was in lower priority for that sprint. I also worked on the release structure for our project.

In the latter half, my role was a tester. I made research on Hudson which is a testing tool for continuous integration. I did some initial configuration to get it run and made some test projects to check out this tool. For stress testing, I did some research on Tsung, I and Jonas discussed on various test environments but finally fixed with some five testing setups. We then got some nice fancy graphs for our project. I also did unit testing for the some of the modules using E-unit and documentation for the our project using Edoc.

A.2 Yury Dorofeev

In the beginning of the project, like everyone else, I was doing research. My area was database and CMS. In database there were three candidates: CouchDB, Riak and Hibari. CouchDB had more advantage over its competitors such as proper documentation, easy way of data storing/retrieving. In CMS there was one candidate only - Zotonic. But it uses PostgresQL which did not satisfy our precondition (we were allowed to use Erlang based tools only). That is why we rejected it.
Then I took part in presentation for bachelor students. We gave them basic information about our course.
After presentation I was doing db API planning. Then I validated forum tool ZincBB. It did not fit our system. After that I switched to evaluate different test methodologies. During the couple of days together with Wilson we learned Common test and Unit test. We wrote instructions on Redmine server for both of them and made a school for other team members.
When the test methodology was done I prepared Redmine school.
In the beginning of October I started working on installation scripts for our system. Then our team was working on APIs implementation. My part was db API.
After API I went back to Common test. Simple test cases were written and we tried to run it on Hudson server. There were some technical problems but we managed to fix them and Common test was successfully run. When Henrik Tallin brought the Crawler and CMS from Erlang Solution I was involved in Crawler evaluation.
At the beginning of November we prepared to Erlang conference. I was responsible for posters.

After the posters were done I wrote start-up scripts for our system's components. Edoc school was my another activity in this project.
When the main parts of the system were done we fixed system's behavior. I was one of those who were working on implementing supervision tree.
I left the supervision implementation and switched to CMS because it had higher priority that time. I was working on implementation of dynamic menu for our web site.
My final business in the project was writing Eunit test cases for different system's components. It took me one and half week. Test coverage was done by 90

A.3 Jacob Ericsson

I was appointed spokesperson for the group basically from the first real work day. This entails keeping the contact with Erlang Solutions, Olle Gällmo and any other person we would need formal contact with.

During the planning phase of the project I worked with checking out forums, wikis, services and databases for our project. After we picked components I spent quite some time familiarizing myself with CouchDB.

Later I worked on one the first connectors we built, the db_connector. I was also a part of constructing the database APIs, both the external as well as internal APIs. During the second sprint I also worked with Wilson to prepare for the demo and make our presentation work.

I defined the coding standard together with Niclas, I extended Ecouch's functionality to allow for changing user and password on the fly and not just during startup, I Wrote test cases for the internal database API, external database API and external Authentication API.

I Revamped and polished the poster that was used at the Erlang User Conference, I refactored all our different connectors into a single gen_connector with a callback module for each of our services. Took nearly a week of work for both me and Jonas.

I also Worked on the slides for the review presentation we had Friday the 17th. Also did half the public talking on said presentation together with Niclas.

A.4 Jonas Rosling

In the beginning of the project I did some research on authentication for users of the website and found e_auth which was integrated into Erlang web and also what message queue to use for the product since we were required to have one of those. The only one that really seemed to fit was RabbitMQ and I also was giving it a test run together with Wilson to get a better understanding of how it really worked and later how to integrate it into our system. I had a interviewing session with two groups of first year students. I was also very involved in discussing how to design the product. During roughly first half of the project I worked mostly on APIs for the RabbitMQ and the database, developing the connectors for each module that we were to connect to our system and also the authentication module. After that I helped some at writing html and Rss and also unit test and bug fixing of the system. Then after a while it got obvious that we needed to make a generic connector instead of the multiple connectors we had from the beginning and I have taken part in that migration of the module specific logic from the connectors and created the gen_connector, shortening the total amount of lines of code by several hundred. When the project headed towards its end I struggled with stress testing the system with the Tsung tool, and after some hard hours of work it finally was delivering results. Finally I worked with finalizing the test cases and some final bug-fixing.

A.5 Yeli Zhu

I started the project by doing some research on the web-frameworks, mainly concentrating on two candidates: Erlang-web and Nitrogen. Later on I did some research on Rebar, Hovercraft,

CouchDB, Ecouch, template language and template engine (e.g. Javascript, wparts, erlyDTL, Django, Dojo Tookit). During the implementation stage, I have spent a lot of time playing with Erlang Web, trying to construct web pages which can handle user inputs and communicate with the database. I also spent some time to get the CouchDB installed successfully. Later I contributed to the development of the authentication, Planet Erlang and RSS services. I was also involved in the implementation of the dynamic menu in CMS. I also did some work in designing database schema, writing and testing installation scripts, and Edoc. I had experience being a Scrum Master. I participated in group discussions and schools. I also attended the Erlang User Conference in Stockholm. I really enjoyed working in this group. I had a great time with the group during fika and playing Table Tennis together.

A.6 Samuel Strand

I started with doing research mainly in the web framework and webserver areas. I also volunteered to be the main system administrator which in the beginning of the project entailed setting up the server used to host the central repository and the Redmine system. I also read up on Git and held a school about it to the rest of the team; throughout the project it was on me to make sure that the repository was sane and to help out when people had problems with Git. As we transitioned into development, I kept working on the web framework system, connecting Erlang Web to the bus and separating out the webserver. Leading up to the EUC I spent a lot of time piecing things together and fixing bugs to have a functioning demo. After the EUC I worked on documenting and testing the web framework and Webserver and tried, together with Niclas, to integrate the CMS into the system. I restructured the system directory, making in comply closer to the OTP standard. I also made releases for the different applications. In the last sprint before Christmas I worked on refactoring parts of the system into sublimeness. During the final sprint I worked on touching up the system and trying to make sure it was possible to hand it over to the company as well as working on the reports.

A.7 Niclas Stensbäck

In the beginning of the project me and Samuel worked a lot on deciding what web framework and web server we would use in our system, as well as leading a lot of discussions about the different architectural options we could choose between. During sprint one and two I spent some time getting familiarised with Erlang Web, Nitrogen, Yaws and the other candidates we had for our components. I also spent a bit of time fixing up Redmine, installing plug-ins and generally educating people on how they should use it. Throughout the project I contributed much to the different design decisions we made, concerning the system architecture, message structures, API designs etc.

I co-wrote the coding standard that we ended up using mainly along with Jacob.

During sprint three I spent quite some time getting the web framework to work as intended, writing what ended up being the mq_api module. We spent some time discussing this as well, different ways and how the API should work. I also did some work on the gen_connector module. In general I spent quite a lot of time refactoring code and making this nicer and more logical after they had been hacked together. Also, I wrote the first version of the yaws connector which made the web server and web framework behave as separate components on the message bus.

I also held an Erlang Web and MVC-school for the group, to get them up to speed on how they work since we had a rather large discussion about which one we would finally choose. I also held a rebar school, which is the build tool we used for the project, showing off the different features and functionalities it has.

During sprint 4 I rewrote the authentication module to be nicer, since its design at the time was rather haphazard. I made it nicer and rewrote the different modules that used it to

reflect the changes. I also implemented an internal API for the authentication module to make it conform with the double abstraction model we were using.

The Hudson machine we made to automate testing (but which in the end went mostly unused) was my domain as well, I set it up and made it fetch the latest commits in the repository, compile and test them. It worked rather well but due to a bug in the Hudson plug-ins to play sounds we never really got it to alarm people when the build broke as we wanted.

During sprint five, when we had the EUC to plan for, I spent a bit of time working on the poster that we were going to have there, writing and fixing the text on the poster along with Yura and Jacob.

I wrote the RSS component, making use of the RSS "library" that Wilson had coded to hand RSS feeds to the planet.

In order to integrate the CMS into the system, I hacked parts of Erlang Web to make it use the MQ when routing requests to different controllers, and also (re-)wrote the e_auth_trapexit and e_db_trapexit modules that were used by the CMS applications (these will be very central to any future Erlang Web applications to be integrated). This entailed changing many things in the database we used and the API's we had written as well.

A.8 Wilson Tuladhar

At the beginning of the project, I did some research on the bus architecture as to which component to use and how to use them since we were supposed to use a message bus architecture. I also did quite a bit of reading on the web-frameworks such as Nitrogen and Erlang Web. I also looked into authentication options such as open id, O_auth and E_auth which was implemented inside Erlang Web. Then after the research phase, I was involved in constructing the initial API for the message bus (RabbitMQ) such that the messages could be sent and received between two clients via the server, first on one PC and then between two different PCs and finally between various components such as initially between the database and the web-framework. I was then involved in making the authentication module for our system and also integrating the database and the authentication module to make them work as a whole. I was also involved in deciding the testing methodology as to how to test our system (Eunit testing and Common testing).

Since the planet site required RSS feed, I worked on implementing the RSS reader for our system.

I also implemented a module which would read through configuration file for the keys supplied and give their respective value associated with them.

After all the components seems to be working well within the system then it was time to beautify the final output. So I did the web design for the website.

I also had an interview with some of the Bachelor's students regarding the Project CS course.

At the end of the project, I started working on the CMS (Content Management System) to make in run individually regardless of our project and with our system as well. I also wrote some test cases for the authentication module and some documentation of it as well.

And for the final sprint of the project, I was focused on writing the product and the project report.

References

[1] Erlang (2011) [Online]. Available from: http://www.erlang.org/. [Accessed 25/02/2011].

[2] Erlang Community Site. (2011) *Trapexit.org*. [Online]. Available from: http://www.trapexit.org/. [Accessed 25/02/2011].

[3] Erlang Solutions Ltd. (2011) [Online]. Available from: http://www.erlang-solutions.com/. [Accessed 25/02/2011].

[4] Erlang Web (2011) [Online]. Available from: http://www.erlang-web.org/. [Accessed 25/02/2011].

[5] Nitrogen (2011) [Online]. Available from: http://nitrogenproject.com/. [Accessed 25/02/2011].

[6] Zotonic (2011) [Online]. Available from: http://zotonic.com/. [Accessed 25/02/2011].

[7] Takeuchi, H. and I. Nonaka. (1986) New Product Development Game *HARVARD BUSINESS REVIEW*, (January), pp.1-10.

[8] OOPSLA (1995). *Object-Oriented Programming Systems, Languages, and Applications*.

[9] Git (2011) [Online]. Available from: http://git-scm.com/. [Accessed 25/02/2011].

[10] Redmind (2011) [Online]. Available from: http://www.redmine.org/. [Accessed 17/02/2011].

[11] Hudson (2011) [Online]. Available from: http://wiki.apache.org/general/Hudson. [Accessed 25/02/2011].

[12] Rebar (2010) *Rebar: Erlang Build Tool*. [Online]. Available from: https://bitbucket.org/basho/rebar/wiki/Home. [Accessed 25/02/2011].

[13] Ericsson AB. (2006-2010) *EDoc Reference Manual*. [Online]. Available from: http://www.erlang.org/doc/apps/edoc/index.html. [Accessed 25/02/2011].

[14] Mickand and Richard Carlsson (2004-2007) *EUnit - a Lightweight Unit Testing Framework for Erlang*. [Online]. Available from: http://svn.process-one.net/contribs/trunk/eunit/doc/overview-summary.html. [Accessed 25/02/2011].

[15] Meck (2011). [Online]. Available from: https://github.com/eproxus/meck. [Accessed 25/02/2011].

[16] Tsung (2011) *Tsung is an open-source multi-protocol distributed load testing tool*. [Online]. Available from: http://tsung.erlang-projects.org/. [Accessed 18/02/2011].

[17] The Apache Software Foundation (2008-2011) *The Apache CouchDB Project*. [Online]. Available from: http://couchdb.apache.org/. [Accessed 25/02/2011].

[18] Scrum Development Methodology (2011) [Online]. Available from: http://www.scrumalliance.org/learn_about_scrum. [Accessed 25/02/2011].

[19] RSS (2011) [Online]. Available from: http://en.wikipedia.org/wiki/RSS. [Accessed 25/02/2011].

[20] EDoc (2011) [Online]. Available from: http://www.erlang.org/doc/apps/edoc/index.html. [Accessed 25/02/2011].

[21] Content Management System (2011) [Online]. Available from: http://en.wikipedia.org/wiki/Web_content_management_system. [Accessed 25/02/2011].