Project CS 2012 Course Report
Uppsala University

Daniele Bacarella
Jon Borglund
Paolo Boschini
Kiril Goguev
Faroogh Hassan
Marcus Ihlar
Alexander Lindholm
Knut Lorenzen
Harold Martínez
Thomas Nordström
Thiago Costa Porto
Linus Sunde
Kim-Anh Tran

Abstract

Project CS is a full-time full semester course where students learn to work in a big project in cooperation with industry partners.

In the 2012 version of the course students collaborated with Ericsson Research to develop applications in the domain of Information Centric Networking. One team of five developed an ICN-enabled web browser for Google's Android platform, while another team of eight developed a backend ICN prototype using Erlang.

Scrum was considered a big help in the development of the products. The groups experienced difficulties with planning and time-estimation of tasks.

Contents

Chapter 1

Introduction

Project CS[1] is a university course at Uppsala University that offers a software project in cooperation with a company. Ericsson Research ([2]) acted as the customer for this project whereas the IT department of Uppsala University facilitated us with a project room, software and hardware infrastructure.

The main purpose of this project was to develop a product based on Information-Centric Networking(ICN), as described in [11]. One out of four architectures that realize this concept of ICN is the Network of Information (NetInf, see [11]). The new idea behind ICN and thus NetInf is to retrieve requested content from any device instead of retrieving it from a specific host. As a result, content can be theoretically downloaded from any number of nearby devices via peer-to-peer. This way network congestion could be avoided.

The project group was divided into two teams where one team was dedicated for developing the front-end of the application whereas the other team was given the task to develop the back-end. The front-end group was called LISA (abbreviation for "Look! I see ants!" which sounds like ICNs) and consisted of five members whereas the back-end team was named ERNI (Erlang NetInf) and was made up of eight individuals.

Starting off with the front-end's loose requirement to use NetInf within an application that can communicate with the back-end's server, the front-end decided to develop a mobile browser called "Elephant". Elephant looks like

[1] `http://www.it.uu.se/edu/course/homepage/projektDV/ht12`

a normal browser from an end-user's perspective. So what differentiates a normal browser from the developed one? Instead of using normal host-based networking, a more information-centric networking approach is used. Many recent papers and research efforts have noted that we should move the Internet away from its current reliance on purely point-to-point primitives to designs that make the Internet more data-oriented or content-centric. [10]

Therefore the browser is a proof-of-concept towards a new networking model where Information Objects are in focus. For the development Ericsson provided Android phones. Thus, Java was the language of choice.

The back-end team developed the Name Resolution Service (NRS, see [11]) to perform the back-end operations using Erlang/OTP[3]. The details of the functionalities developed by the front-end and back-end teams can be found in the product report.

Both teams followed Scrum as the software development methodology. Scrum is a framework structured to support complex product development. Scrum consists of Scrum Teams and their associated roles, events, artifacts, and rules ([15]). You can find a short description about Scrum and how we used Scrum in Section 3.1.

This document describes in detail the various tools, methodology and equipment we used to achieve our goals. For more technical details about the products developed during the course please refer to the product report[12].

Chapter 2

Resources

2.1 Project Group

In our project we were 13 students coming from 7 different countries (see Figure 2.1). It was a great experience to have team mates from different parts of the globe - not only working-wise but also food-wise! As our fika rule states, any person who comes later than 9 o'clock should bring fika for the rest of the group. Thus, we could try Tequeños ("cheese sticks") from Venezuela or Tiramisu from Italy.

Due the scale of the project, the group decided to divide itself into two teams: The frontend team (LISA), responsible for implementing the client side of the NetInf project and the backend team (ERNI), responsible for implementing the server technology.

The groups were divided as follows:

Group member	Nationality
Daniele Bacarella	
Jon Borglund	
Paolo Boschini	
Kiril Goguev	
Faroogh Hassan	
Marcus Ihlar	
Alexander Lindholm	
Knut Lorenzen	
Harold Martínez	
Thomas Nordström	
Thiago Costa Porto	
Linus Sunde	
Kim-Anh Tran	

Table 2.1: Each group member's nationality

The Frontend team (LISA)
Paolo Boschini
Harold Martínez
Thiago Costa Porto
Linus Sunde
Kim-Anh Tran

The Backend team (ERNI)
Daniele Bacarella
Jon Borglund
Kiril Goguev
Faroogh Hassan
Alexander Lindholm
Knut Lorenzen
Thomas Nordström

Since we had two office rooms, each team could have one of its own. Nevertheless, we had to continuously communicate with each other, since we were working on the same project with the same customer.

2.1.1 Seating arrangements

The seating arrangement of both groups are shown in Figure 2.1 and Figure 2.2. The main point of the seating within both groups was to face each other, so that social interaction and communication was facilitated.

For bigger discussions though, the backend team used the coffee room, whereas the frontend team had a separate discussion table in the corner

Figure 2.1: Frontend seating arrangement

of the room. Therefore other team mates could continue working without being distracted.

2.2 Equipment

All the members in the teams were provided with a Desktop PC. In addition to the main development PC's there were three servers running Git, Redmine, JIRA and Jenkins and a projector from Uppsala University. The teams installed Ubuntu 12.04 LTS on all the machines.

Ericsson Research also gave the LISA team eight Android phones for development and testing. Each Android phone came with the JellyBean Android OS.

Figure 2.2: Backend seating arrangement

2.3 Tools

2.3.1 Development Languages

The LISA team used Java along with the Android SDK. Their application was targeted at JellyBean Android OS version 1.4.1 (Version 16).

The ERNI team used Erlang, Javascript, and HTML 5 as their development languages. The main product was coded in Erlang. During the final sprints the client wanted to add video streaming. As a result of this ERNI also created an HTML client interface to our system using Javascript and HTML 5.

2.3.2 Continuous Integration & Build Server

A continuous integration (CI) server was adopted which allowed the developers to create special 'jobs' which controlled the compilation, error reporting/blaming and source control. Other instances of this project used tools like Buildbot[1]. but after a long and fruitless effort to make and configure buildbot properly the teams determined that Buildbot was a waste of time and looked into a more powerful/user friendly CI server called Jenkins[6].

Jenkins is a CI tool with an HTTP interface that allows users to setup custom jobs for their project. Things like monitoring a version control system for changes or running command line arguments on the code in the project is easy and user friendly with Jenkins due to plenty of tutorials and resources available online. Jenkins controlled both the ERNI and LISA project.

Jenkins was particularly useful for informing individuals who 'broke the build'. If Jenkins is configured with the email addresses of all people and a build job fails after someone has updated the source code then, Jenkins will send an email out notifying those who have broken the build and those who are affected (all other collaborators on the current code file which is broken).

Appendix C C.1 describes in detail how Jenkins was setup for use in this project.

2.3.3 Version Control

In order to keep the workflow going at a good pace the teams elected to have version control, which is good practice in all projects. Git[5] was used with a custom workflow shown below.

Have a server side repository with 4 initial persistent branches.

- master
- staging
- develop
- release

The following naming convention for temporary branches was adopted:

- SprintX.shortStoryName

The temporary branches were deleted after each successful merge to the DEVELOP branch.

2.3.4 Policies

- master

  The 'master' branch was allowed only to contain Demo code. This is the code which contains ONLY the fully tested and integrated stories.

  Tags were made here under the following convention:
  SprintX.shortStoryName

  This was Jenkins build tool controlled area - No human user was allowed to operate in this branch.
  Jenkins is responsible for merging from 'release' to 'master' at the end of a sprint- in order to keep the branches synchronized and provide a fresh clean start for each sprint from working demo code.

- release

  The 'release' branch was allowed only to contain individual stories which were completed and fully unit tested. Here the team could pick and choose which stories to include in a specific demo. This branch was also a Jenkins build tool area.
  Jenkins was responsible for integration testing and merging between 'release' and 'master'.

- develop

  The 'develop' branch was allowed to contain all the code that was able to be compiled on the server and is where the human users would start their personal story branches. Also a Jenkins build tool area, the code here was considered in a "Story done and compiles but not yet tested" state.
  Jenkins was responsible for unit testing and merging between 'develop' and 'release'.

- staging

  The 'staging' branch contained all the dirty code and is where the human users would push all their code when finished for the end of

the day. This was also the branch where Jenkins was used. Jenkins would pull each new commit and try to compile it, if it compiled then it would be merged with the 'develop' branch. If not, Jenkins would notify the users who had commited changes since the last failing commit.

- SprintX.shortStoryName

  The branch's name contained the word 'Sprint' with the current sprint number appended by a short story name-typically the name written on the post-it note for example: Message_Handler. A merge to the 'develop' branch would mean the story was considered done for the sprint but required testing by integration tools and Jenkins. This branch would be deleted after the tests were passed and a successful merge was complete.

For instructions on how to utilize this workflow see Appendix B.0.1.

2.4 Project Management

There have been many suggestions from previous instances of this course to use project management/issue trackers. Initially both teams were using Redmine[8] but then decided to diverge and try out another project management tool.

2.4.1 Redmine

The ERNI team decided to use Redmine[8] for the duration of the project. The tool provided useful features such as

- Wiki
- Version Control explorer
- Bug and Issue tracking
- Time keeping
- File storage

Redmine is easy to install and free of cost, including plenty of plugins for various needs. BitNami offers a free one-click-installer package for Redmine. Both teams started the first sprint with Redmine installed and two projects configured. Redmine was used heavily for the first sprint by ERNI for creating issues and trying to keep track of time spent for each task in order to aid in planning the time required for each task.

However by sprint two's retrospective and a changeover to a new Scrum master, the ERNI team decided to drop the issue tracking and the time keeping as they felt it was too much overhead and did not help so much with the time estimation of tasks. The wiki became the most extensively used feature of this tool followed by the Version control explorer.

Instead of creating issues and assigning them in Redmine the ERNI team opted for a simpler solution, writing tasks on post-it notes and adding them to the Scrum board. Even though this was simpler, it became messier as sprints became longer.

2.4.2 JIRA

The LISA team wanted to test out JIRA[7] since they had many issues getting the plugins and the right functionality for their team. JIRA unlike Redmine is not free, and comes with a 3 month trial license. LISA felt that this provided much more functionality than Redmine and was easily customizable.

JIRA worked well for LISA as each member was at one point Scrum Master, and was able to keep work logs, issues and assign tasks easily. The major point for using JIRA was persistence as each Scrum sprint was catalogued.

Chapter 3

Project methodology and organization

3.1 Scrum

Scrum is an agile software development framework that is used for planning, managing and undertaking a software project. Scrum is designed to optimize flexibility and productivity, as it demands short working phases (sprints) that make it possible to deliver a working software product continuously throughout the project. Thus, the Scrum team can react on time when the customer changes the product specifications.

Agile methodologies are rapidly becoming the standard in software development companies. As a student, the Project CS is a great opportunity to experience Scrum, as we are about to go into the real world of software development.

3.1.1 Roles

Scrum distinguishes between the Scrum Master, the Product Owner and the Development Team.

The Scrum Master acts as a bridge between the team and the product owner. He is responsible for making sure that the team adhere to the Scrum guidelines. He is not to be seen as the head of the group, but instead as

someone who's main responsibility is to remove impediments in completing a task. He should also make sure that the team is on the right track keeping a constant eye on the sprint goal and the definition of "done" for the different tasks. On the other hand, the Scrum Master needs to ensure that nothing interferes with the development team so that the developers can constantly concentrate on their work.

The Product Owner is an individual who can be seen as a connection between the client and the development team. Within the team, the Product Owner acts as the client, although the client normally is an external actor. The Product Owner is the person who owns and controls the development of the software with the help of the backlog, prioritizing features and generally the one who is supposed to give direction to the development team during the sprint planning and at the product demonstrations. In some cases, the Product Owner can be the same person as the client.

The Development Team is the set of individuals who are working on the project for the product owner. The team is responsible for creating increments and releasing a working version of the software after each sprint.

3.1.2 Scrum Keywords

- **User story**
  A user story is a description of an end user interacting with one small part of the product. It gives the developers an intuition of what functionalities need to be available to accomplish the intention of the user. Each product requirement is translated into a user story. Stories are usually broken down into tasks and each task is the smallest unit of work to be implemented. It is important that the development team agrees on the definition of done for each task and story.

- **Product backlog**
  The product backlog is the ordered list of all user stories for a product. The backlog is created by the customer. Stories are prioritized by the product owner depending on criteria such as date needed or business value.

- **Sprint**
  A sprint is the core artifact of Scrum. A sprint is an iteration that usually lasts between two to four weeks during which a part of an entire product is implemented. Each sprint has a goal and each member of

the team should strive to fulfill that goal. The whole project period consists of several sprints.

- **Sprint planning**
  In the beginning of each sprint the team meets for planning and agreeing on the current sprint's goal. Stories are picked from the product backlog and broken down into tasks. The team estimates the workload for each task and moves the highest prioritized tasks from the product backlog to the sprint backlog. This is done until the maximum workload is reached for the sprint. The output of the sprint planning is a sprint backlog that everyone agrees on.

- **Sprint backlog**
  A list of stories that should be completed by the end of the sprint. The sprint backlog is filled by picking the highest prioritized stories one by one from the product backlog.

- **Sprint demo**
  At the end of each sprint a demo is scheduled with the client. At the demo the team presents the results of the sprint in the form of a working product to the customer. The good part of having frequent demos is that the team and the customer can feel the progress. It is also an opportunity to see if the customer and developers share the same idea.

- **Sprint retrospective**
  After each sprint is completed, the team gathers together to reflect on the good and bad parts of the sprint. To make improvements throughout the project it is important that everyone is honest and shares their opinion during the retrospective. This is the point where you can not only improve the working process but also the team environment.

- **Standup meetings**
  A mandatory short meeting (usually 10 minutes) starting every day at the same time. Each person tells the others what he did the day before, what he is going to do next and if there are any impediments (problems) preventing him from finishing his tasks. The participants attend the meeting standing up so that everyone is encouraged to be concise when telling the status of their work.

3.1.3 Scrum process

The following picture summarizes the Scrum methodology.

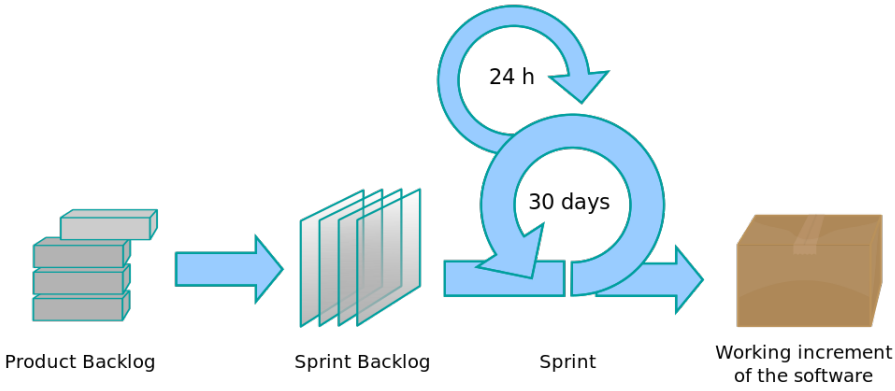

Figure 3.1: The Scrum Process[**?**]
`http://effectiveagiledev.com/portals/0/scrum-process-basic.png`

3.2 Use of Scrum in this project

Throughout the project the teams tried to apply the Scrum methodologies thoroughly. Both rooms and desks were arranged to facilitate direct communication among team mates, as well as setting up a whiteboard for the post-its. The frontend team agreed that each team member should try to be the Scrum Master at least once during the whole duration of the project. Project CS provides a great advantage in that it is a course within university and the Scrum master role could easily be shifted from one person to the next.. Since there were two teams, there was always two Scrum Masters, one for each team.

3.2.1 Daily meetings

Due to the separation of the group into two teams, there were two distinct sets of meetings in the offices. Daily meetings for the frontend team took

place everyday at 9:00 o'clock sharp and time boxed to 15 minutes, while at 9:15 the standup meeting would start for the backend group. Both scrum masters were present at each meeting. The reason for this was to facilitate synchronization between both teams.

In our opinion these meetings were of great value. They encouraged everyone to actively talk to the other team members about the status and the problems they faced while completing a task. Each stand up meeting was mandatory so that we could be sure that everyone would start working at the time. Moreover it was a good way for planning short tasks and catching bad decisions early.

3.2.2 Sprint planning

After each demo the teams planned the sprint for the next iteration, trying to get done before the weekend. In this way work started directly at the beginning of the new week. Most of the times the demos were on Thursdays. Thursday was chosen so that pre-planning could be done on the same day and to meet the client on Friday for planning approval.

To estimate the available working days for each sprint, the teams calculated the total amount of days according to the number of the team members, and used a productivity factor of 0.7. One story point corresponds to one man day (8 hours). A team of 5 developers working for two weeks would have 5*10*0.7 = 35 available story points. In order to estimate the workload of each story "planning poker" was used. Each individual had a set of cards with numbers and votes with a card giving points to each story.

In our case, the requirements were not set from the beginning. Therefore the teams had a continuously changing product backlog, which was updated at each pre-planning phase.

3.2.3 Demo

Demos were held at the end of each sprint during which the teams showed their results. We demonstrated the application to Olle, Muneeb and Ericsson in a very relaxed environment. Having short sprints and doing frequent demos ensured that Ericsson saw progress often.

3.2.4 Retrospectives

Retrospectives were done in form of meetings after each sprint. Each member had to say three good things and three bad things about the completed sprint. It was a great way to point out what was positive during the sprint and to have constructive criticism for the bad parts. In this way the teams could always try to improve the process in the next sprint.

3.2.5 Conflicts

In any large group project there will be many different personalities, opinions, experiences and backgrounds. Some people have very little experience of working in larger groups (10+ people). This course forces the members to work together and to do it in a specific project management style (Scrum). It is inevitable that there will be conflicts and disagreements.

The following are some points that the teams followed when there were conflicts.

- Try to address the problem with the individuals involved.

  The first and foremost approach is to try to talk it out with the people involved. Please note that you may have very different personal and work experiences. This is a good thing but keep in mind that how you worked in a previous course or at a previous workplace may not be the best way to do it in the current project. It is best to come to the project with an open mind and offer your past experiences as a way to suggest how to go about preforming certain tasks, but never blatantly ignore everyone else because you feel they are wrong. This course is a team-effort based project and not to be treated as a workplace, here you are allowed and even encouraged to make mistakes and to learn from them.

- Realize that just because you do a task does not mean that it was the right one.

  The idea of Scrum, simply put, is to change the product during the development cycle quickly. As developers we may sometimes get attached to the code we have written and can take offence when the

team or client decides to remove the code. In this case it is not a matter of the team or client attacking you, it is a matter of whether or not the code or task is still relevant to the project after a design change. The project will be very different from the beginning to the end. Code will be obsolete and deprecated at certain points, just live with it and continue contributing and understand that your work is still recognized by everyone else.

- Use the Scrum master as a mediator.

  In the case where addressing the problem yourself did not work out, the use of Scrum in this project gives everyone a great resource, the Scrum Master. It is the Scrum Masters duty to remove impediments in the sprint, however big or small. Conflicts are a great impediment and can sometimes lead to failed sprints and worse, de-motivation of the team. The Scrum Master should in this case act as an impartial mediator. And strive to come up with a solution to the problem, even if it means making sure the conflicting personalities do not work together.

3.3 Quality Assurance

In large projects there is a need to be able to control the quality of the product. Does it do what the client asked? Is it extensible? Is it releasable? While some companies have specific quality assurance teams, this course usually does not. Therefore, the team had to explore ways of answering the above questions.

While Scrum and Agile methodology easily answers the "Is it releasable?" question, just because the team implements Scrum and Agile does not mean it is being followed correctly. In Scrum, at the end of every sprint the team should have something that is demo-able. Sometimes this is not the case – failed sprints, outstanding issues, etc.

3.3.1 Pair-programming

Both teams decided to take the pair programming approach for quality assurance. Two programmers complete a task and two other programmers review that work for bugs, imperfections (according to coding standards), and integration status.

The ERNI team felt that this worked particularly well as we all had little to no experience in functional programming and large group work projects. Often bugs were caught way before the task was completed and put into the review process.

Using pair programming also helped develop our skills faster as we taught each other how the language worked and had knowledge redundancy (if one person was away the task would not be stopped). After review you had twice as many members of the team having full knowledge of the code and functionality introduced into the product.

3.3.2 Code Reviews

The LISA team decided to use a tool called Gerrit[4] to preform code reviews. Gerrit is a web-based code review system, which makes it easier for us to do the review, as every team member can see what is up for review and what has been reviewed at all times. Gerrit is dependent on Git and acts almost like a filter, only allowing authorized changes to be pushed to the master branch on Git. Therefore, it promotes code quality by forcing code reviews to take place before code is placed on the repository.

Initially, we had some trouble setting up the tool. While installing Gerrit itself is not a problem, integrating it with Jenkins was a bit of an issue. After a few emails with the maker of the Gerrit/Jenkins plugin, we found out that the issue was due to encoding in the terminal. Make sure you have the proper terminal encoding (UTF-8) before setting it up. After this small mix-up, Gerrit was ready to be used and tests suggested that it was working properly.

After everything was running, it took us about 2 hours to drop Gerrit altogether. The reason for it was mainly cultural – we implemented a code reviewing tool in the middle of the project – and we did not have enough time at that point to invest in learning how to do it properly and maybe

circumvent the main issue we had with the tool. The main issue was that, in our team of 5, code was flying around back and forth. New feature branches were created and merged into the main branch. These branches often contained features that were required by other features that were being developed at the same time. Gerrit provided a filter, as advertised, but it ended up with not being very productive for us. We would have to change our workflow and, in the middle of the project, that was a major overhead for all of us. Being a team of 5 people, we could manage, to a certain degree, the code quality by doing pair programming and reporting to others about strange code that one has come across.

Gerrit is a very nice and interesting tool, which will probably help your development a lot *if* you implement it before the actual coding has begun. Thrive to understand and make effective use of this tool. And follow the proper installation guides.

3.4 Timeline

This section presents a timeline of both team's Scrum sprints. It is recommended that the timeline is read after or in conjunction with the product report[12], as it details how our final product was implemented. The timeline specifically points out the technical aspects that were tackled during each sprint.

Both teams were synchronized with each other until sprint 4 when the ERNI group chose to have two one week sprints instead of one two week sprint. The result was a series of one week sprints for the ERNI team ending with seven sprints in total, while LISA had only six.

The course ran over a period of four and a half months starting on September the 3rd and ended on January 18th, with two major presentations. This particular instance of the course was unique since the client did not have a very concrete project laid out in terms of requirements. In the beginning of the course, a lot of time was spent learning about ICN and thinking about scenarios which would be interesting to implement. We held different meetings on trying to unders Therefore the real work and Scrum sprints started slightly later than other course instances. Furthermore there were too few people in order to have two groups with two completely different clients.

3.4.1 Pre-Scrum

During the month of September the group met with the client who presented the idea, although it was not very concrete at the time. The basis for the project was known – the group would work with Information-Centric Networks – but what was going to be developed was still undecided.

This meant that the group had the freedom to discuss different scenarios to be implemented during the course. It also meant that the first few weeks of the course were spent reading up on research papers that discussed ICN. There were numerous meetings during September that helped increase the knowledge about ICN and investigate further the pros and cons of this technology.

In previous years, the course started with an Erlang workshop which consists of two weeks of intensive Erlang. However, this instance of the project was so open, that it was unknown whether the project would need Erlang at all.

By the end of the month, we had a better idea of what we were going to do, especially after meeting our client. It was also time to separate the groups, define the teams and rearrange our workspace. All of this happened right before the first sprint, which started after approximately one month.

3.4.2 LISA

Sprint 1	
Scrum Master	Thiago
Goal	Send and receive a message to the backend's server
Sprint length	2 weeks
Met	Yes

Sprint 1 started off with setting up the working environment. This included reading up on and configuring the tools we chose to use during the project, which are described in Section 2.3. The work of installing the tools was directed to the ERNI group, which took over the tools for the beginning of the project.

A major point for this sprint was also getting started with Scrum. It was very important for the team to develop a culture, some sort of a Scrum

routine that we were going to follow. The front-end did a good job doing this from the start of the project.

During the initial stages of development, the front-end team had a Skype meeting with Hugo Negrette Otaola and Miguel Sosa, who had worked on a similar project before. They wrote a Master Thesis [13] that was a fundamental starting point for the project.

Finally, the front-end implemented a simple application that could send a message to the server developed by the Backend team and receive an answer. The front-end turned out to have more time than expected, so parts of the Bluetooth communication and some side functions were implemented.

Sprint 2	
Scrum Master	Thiago
Goal	Communicate with another device using Bluetooth & Publish and retrieve content with metadata
Sprint length	2 weeks
Met	Yes

Now that we succeeded to communicate with the backend's server, we needed to send messages according to the NetInf specifications in [14]. Thus, we investigated the specifications and implemented those message regulations into our code.

We designed our first architecture draft based on OpenNetInf and implemented the most important modules for sending/receiving messages to/from the NRS - this time using OpenNetInf. We managed to share objects between Android phones using Bluetooth.

At this stage, the application could request content using a short hash. The specified hash was sent to the NRS and a list of locators, that owned the requested object, were recieved in return. The locators were Bluetooth MAC adresses. The application was then starting a Bluetooth discovery in order to find out whether one of these locators was within reach. If so, a Bluetooth communication to that locator was established. The hash was sent to the connected device, which replied with the file that was identified by the hash.

The efforts at the end of the sprint were focused on the message that was being shared and we spend two days creating the objects and making sure the under-the-hood part of our application was working. During this time, we

worked on creating the Metadata and a parser for it, building the messages in a correct manner and checking where files were being saved and retrieved from. We also worked on getting multiple types of files available for visualization, leading us to work with the Gallery supported by the Android phones. We ended up using the files' mime-type to open them.

Sprint 3	
Scrum Master	Paolo
Goal	Successful presentation. Search content, cache and retrieve content from the NRS, implement a minimal web interface
Sprint length	3 weeks
Met	Yes

Within the first week we mainly worked on the presentation for our review at Ericsson in Kista. We prepared a paper prototype (consisting of screen shots of the final application) of our project idea of creating a browser application that we later called Elephant. The application looks like a normal browser and behaves as expected. The idea that we wanted to convey was that the browser uses information-based networking instead of location-based networking. Our idea was to create a browser using NetInf in a way, that the user does not need to know what actually happens. It would be a first step of changing our way of networking. *Note:* Using the paper prototype helped us a lot to agree on our project in the end. Our client could understand our ideas and was very happy to see where the project was going.

Before continuing adding new functionalities, we took some time during the third sprint to clean up and refactor the code. The git branches were restructured due to the experience gained from the previous sprints. The structure is described in Section 2.3.3.

The new functionalities we added were publishing (register the device as a locator), full-put (uploading the content to the NRS) and searching for contents using URLs. Furthermore we added a Local Resolution Service (LRS) besides the NRS. The LRS looked up content in a local database, that we designed and implemented within this sprint.

Now our application could search for content within a local database and in a remote NRS by a URL. The response was a corresponding hash, that identified a web page associated to that URL. In case the NRS owned the web page that was searched for, we directly retrieved that web page instead

of a hash. Using the hash, we could get a web page from the LRS or a list of locators from the NRS. In case of a list of locators, the device would start to connect to other devices and download the content through Bluetooth. As soon as we retrieved the web page, we could register ourself as a locator in the NRS or even upload the content to the NRS. At this stage we displayed simple HTML web pages within a Android WebView environment, without any pictures or java scripts.

It is worth to note that the search implemented in this stage was not developed in accordance with the NetInf standards, not being connected to Resolution controllers. This was due to a perceived complexity of the code and the short amount of time we had to finish the sprint. We left this to be completed in the next sprint.

Sprint 4	
Scrum Master	Kim-Anh
Goal	Higher granularity browsing
Sprint length	2 weeks
Met	No

As requested from Ericsson, we separated our application into two applications: Elephant, our browser that uses the services of another application and the NetInf services.

In sprint 4 we created our final application. This included creating the minimal browsing functionalities: Handling clicks on links as well as displaying web pages correctly as they are displayed in other browsers. The main challenge was to intercept the Android WebView to redirect resource requests to our NetInf services, instead of simply downloading them from uplink.

We also integrated the search functionality to NetInf, making use of the Search Controller and implementing our search functionalities as Search Services.

Furthermore we needed to add settings and help pages. The user should be able to decide which NRS to connect to and whether he wants to upload content to the NRS or register his device as a locator.

After sprint 4 we had our final application that offered a browsing functionality based on NetInf. Since some bugs were left to be reviewed, we tackled these in the upcoming sprint.

Sprint 5	
Scrum Master	Linus
Goal	Running application without bugs (not necessarily with Bluetooth Convergence Layer)
Sprint length	2 weeks
Met	Yes

During this sprint we tried to fix a bug that prevented some pages to display correctly. The bug showed up during the previous demo, so the goal of this sprint was to solve this bug and make sure we had a working application. After several tests we spotted the problem that was causing the corrupted pages. The problem resided in the way we managed the search function and the messages sent to and from the NRS, causing the loading of the wrong identifier of the sought web page after a search. The backend could solve the problem easily and our application started working as expected. During this iteration we also noticed some slow downs due to how we used the timeout in our network requests, so we fine tuned them to get a decent speed when browsing web pages.

Afterwards we discussed what to evaluate and how to evaluate our application. We then implemented logging of network activities by storing each request with the relative type of transmission used (uplink, NRS cache, bluetooth or database). In order to produce some statistics we also stored the amount of data that is transferred for each request and the time that it takes to transfer the data. To simplify the test we added a functionality to automate the loading of several web pages in sequence. That way it became possible to gather a considerable amount of data from many web pages during each run of the application.

At this point we had a working web browser that used NetInf services and could support four different types of transmission link. The next and last step would be to run the automated tests and analyze the obtained results, so that hopefully we would have interesting material to show in the final report.

Sprint 6	
Scrum Master	Harold
Goal	Successful presentation and both the product and course report
Sprint length	2 weeks
Met	Yes

The last sprint involved the preparation of the final presentation and writing the final course report and product report. As usual we divided the work into different tasks so that we could write different parts of both reports simultaneously. The difference between the course report and the project report is the target reader. The intended audience of the course report are the teachers and future students of this course, while the product report is intended for the customer. The course report contains information on how we got to the final product, while the product report focuses more on the product itself, the implementation, how to run it and how to continue the development.

In this sprint we also ran the automated tests we had implemented in the previous sprint and organized the results. The final presentation was made with Prezi[1], an online tool to produce presentations.

3.4.3 ERNI

Sprint 1	
Scrum Master	Knut
Goal	Send and receive a message to the frontend's client
Sprint length	2 weeks
Met	Yes

During the first two-week sprint the ERNI team mainly setup their environment along with coordinating with LISA on building a server which would communicate to a client using simple non NetInf messages. The code from this demo became the reference for the real NetInf server which was developed during the second sprint.

[1] `http://prezi.com/`

Sprint 2	
Scrum Master	Knut
Goal	Implement NetInf Publish and Get content with metadata
Sprint length	2 weeks
Met	Yes

In the second sprint which ran for another two weeks, ERNI created the basic NetInf server which was able to use the appropriate Publish and Get messages from the NetInf protocol draft. A major part of the sprint was devoted to creating the Message handler and the formatter which would accept messages passed in from the HTTP handler and convert them to the internal representation of a NetInf message. The ERNI team also developed a very basic list database in order to store information.

Sprint 3	
Scrum Master	Faroogh
Goal	Implement NetInf Search, binary caching and retrieve content, implement a real database (Riak) and implement a plug and play database wrapper
Sprint length	3 weeks
Met	Yes

The third sprint consisted of refining the Publish and Get messages, implementing search and binary storage and finally introducing a real database. The client also stressed that they would like the ability to easily attach other databases, thus a database interface was created and the ERNI team attached Riak[9] to the NetInf system. During this sprint a presentation of the progress so far was held at Ericsson in Kista.

Sprint 4	
Scrum Master	Jon
Goal	Provide the client with a draft of the NetInf Video Streaming
Sprint length	1 week
Met	No

During the fourth sprint the ERNI team had chosen to diverge from the

LISA team since we had finished all the functionality described in the draft document and were no longer required to build features for LISA. The client was impressed and suggested that the team look into applications for the NetInf system. The major application that was discussed was the use of NetInf and Video streaming to alleviate congestion problems while broadcasting content. The goal here was to provide the client with a draft of the new protocol but there were many disagreements on what the draft should contain and the team missed the deadline to hand in the draft.

In an effort to fine tune our sprint planning, the team decided to try one week sprints in order to see how much we could accomplish with a smaller amount of time. This was due to problems estimating the amount of work the team could finish. The team always overestimated the time it would take to finish tasks leaving the team with too little to do at the end of sprints.

Sprint 5	
Scrum Master	Jon
Goal	Implement NetInf video streaming on top of the existing NetInf NRS code and UDP convergence layer
Sprint length	1 week
Met	Yes

In the fifth sprint the ERNI team had gotten the approval of the client to start implementing the NetInf video streaming protocol. It was a one week sprint where the team also added the UDP convergence layer after request from the client in the previous sprint.

Sprint 6	
Scrum Master	Marcus
Goal	Fixing bugs within the system, implementing feedback from client on video streaming
Sprint length	2 weeks
Met	Yes

In the sixth sprint the ERNI team mostly fixed the main bugs in the system and finalizing the NetInf video streaming. The major feedback here was to get rid of the HTTP content-dispatcher and replace it with a modified version that uses NetInf messages to transfer the chunks.

In the final sprint both teams joined together and had one large sprint

writing the course and product reports.

Chapter 4

Team Building

4.1 Team Building

In order to maintain a friendly environment at the work place and to make people feel comfortable and confident with each other, the team arranged a series of activities which are explained in the upcoming sections.

4.1.1 Fika

Fika is a Swedish word that is used to describe a break to drink coffee or tea. It was a popular word in our project. Every Wednesday two group members would bring something to share with the rest of the group. The fika break started at 15.00 on Wednesdays. This time to get to know each other better and discussed everything from politics to weather. Apart from the weekly fika, there was also the "late arrival punishment fika", where anyone who didn't show up on time would arrange an extra fika.

4.1.2 Birthdays

Group members who had a birthday during the course were celebrated. For this purpose money was collected at the start of the project from everyone. This "birthday fund" was used to buy a cake for the birthday celebration. The teams would then take a break from work to celebrate the birthday of

the person and eat the cake with coffee or tea.

4.1.3 Eating out

Every now and then the team used to go to a restaurant or a student nation to eat something for lunch or dinner. Also when the weather was nice at the beginning of the project, there was an outdoor BBQ which gave the team members the opportunity to spend time with each other outside of the working environment.

4.1.4 Bowling

After the midcourse presentation at Ericsson Research in Kista, they invited us to bowl, drink beer and eat dinner at a bowling alley in Stockholm. This was a really good evening, since the students got a chance to mingle with the people involved in the project on a more personal level.

Chapter 5

Conclusion

5.1 Conclusion

At the end of this project both teams can proudly say that this project was a success. Almost all the goals that we had set for ourselves at the beginning of the project and during each sprint were achieved and those which were not were achieved later on. The client, Ericsson Research, was happy with the performance and offered thesis opportunities to a number of team members to continue working on the application that was built.

- **What went well:**
  Even though the teams were not 100% efficient, Scrum helped a lot during all stages of this project. The daily stand-up meeting kept everyone updated on the progress of the project. The sprint planning gave everyone the opportunity to participate in meaningful discussions regarding how to build different functionalities into the product. A lot of feedback was gained during the retrospectives that made it possible to improve the product development. The teams are thankful to the teaching staff for arranging different workhops for us. The Agile, Erlang and Testing workshops, held by experienced professionals who have been part of the industry for many years, were very helpful.

  For those who are going to be working with Erlang in future instances of this course, Erlang OTP in action is a great resource and helped in the beginning of the course. Traditionally the course has a two week Erlang workshop, but this instance had one week of self-study

(Erlang OTP In Action and the online resource "learn you some erlang for great good"[1]) followed by two days of an Erlang workshop. The teams believe this was a much better way to handle things since the opportunity to make all the mistakes before and ask valuable questions afterwards when the expert came was there.

The weekly fika is also something that is recommended to future students. It is a good way to keep a friendly environment in the team and to get to know each other.

- **What did not go well:**
  Another thing learnt during the course of this project was that not everything is going to be the way it is planned to be. There were problems with estimating the duration of different tasks in almost all the sprints. At times the teams overestimated the time assigned for the completion of a particular task and at times it was underestimated.

  Another problem was choosing the right software tool for a particular purpose. At the beginning of this project a considerable amount of time was spent on installing and reading about the tools that were never used later because of better options that we did not know about in advance. To give an example, the team installed Buildbot for continuous integration but found it difficult to learn and manage so a switch to Jenkins was made instead. The advice to future students is to spend some time in investigating what is the best tool that is easy to use and can be learnt quickly.

- **What can be improved:**
  At times during this project functionalities were built that had to be scrapped later because they were not within the scope of the project. It is important to have a discussion before starting to code anything and to have the consensus of the team on the overall design, but not on every implementation. Communication is very important and team members should not shy away from asking questions or demanding clarifications on anything.

  Communication with client is also an important part of any software project and the teams think that for future instances of this course the client should be involved more in the project and should provide concrete requirements. In our case Ericsson Research was the official

[1] http://www.Learnyousomeerlangforgreatgood.com

client but most of the time the teams had to act as a client for themselves and make decisions that otherwise the real client would have to make.

In closing, always make sure that a working agreement exists (timing, working hours, fika rules and much more) and is kept by all team members! It is important to stick to what everyone has agreed on or else conflicts might arise. Remember to commit to the team. Project CS can be a course which provides the chance to get to know new friends but it also comes with a big responsibility. It is a great opportunity for an individual to work in such a big group since it resembles the work environment in the future.

Finally, readers interested in the individual input by team members can be found in appendix A below. Tool setup guides for a Git workflow, Jenkins Continuous Integration Server as well as Coding Standards can be found in appendices B, C and **??** respectively.

Appendix A

Individual input

A.1 Daniele Bacarella

Sprint 1: I setup my working environment by installing all the software needed to start developing such as the Erlang compiler along with the build tool Rebar and the editor Emacs. Once the environment was ready I started studying and practicing Erlang while getting familiar with the editor Emacs.

Sprint 2: Kiril and I both worked on the NRS logging and the HTTP handler for the system. Then we researched GIT practices and came up with our own workflow and taught it to the group. Afterwards I created the inital version of the project Makefile and integrated Rebar into it.

Sprint 3: Jon and I researched database alternatives to the one already adopted that uses an Erlang internal data structure which is a list of key-value pairs. For obvious reasons it did not represent a valid solution to store data since it did not provide reliability and persistency along with other concurrent features that regular DBMS provide. Among the many available options, we chose Riak. After having set everything up to make it fully working we proceeded writing some tests and the database wrapper for it. Finally we wrote a install guide on the wiki.

Sprint 4: The back-end team started working independently from the front-end team on the new NetInf NRS Streaming feature. It required us to implement an HTTP client interface to the upcoming NetInf Nrs Streaming. During this sprint I worked with Alex to create the first prototype of

the web interface and the Http client which would communicate with the running NRS to perform the operation requests issued by the user. We also researched video players suitable for the web page.

Sprint 5: I worked with Alex and Jon fixing bugs and design in the Http client and the web interface.

Sprint 6: I helped Kiril with starting the Product and Course reports as well as finalizing the HTTP client interface with Alex.

Sprint 7: I started writing the Product and Course reports.

A.2 Jon Borglund

During the first sprint I mainly refreshed my functional programming skills and also studied quite a bit of Erlang OTP. During the first couple of days Faroogh and I compiled some coding standards, and researched Extreme Programming practices which we later presented to the group.

In the second sprint we started the implementation of the NRS. During the first days Alex, Linus and I read and compiled a compact version of the NetInf protocol draft to be sure that both the frontend and the backend would interpret it in the same way. Alex and I also created a curl-script that allowed us to test our NRS according to the draft. We then continued to design, implement and test the initial storage module. We also started to implement an integration test with Erlang and Eunit.

In sprint 3 Thomas and I fixed some bugs in the message handler and message formatter. Then I joined Daniele to choose and add a persistent database. We first investigated which database alternatives there were. We decided to go with Riak, hence we proceeded with its installation, configuration and finally testing.

During sprint 4 I was the Scrum master, therefore I spent time on Scrum master specific tasks, such as entering stuff into Redmine, updating the whiteboard and burn down chart and also conducted minor conflict mediation. Thomas and I defined and implemented interesting state statistics, such as number of active request, number of received request etc. We also went through the NetInf protocol draft again to be sure that we have covered everything in the HTTP convergence layer in our implementation. Marcus, Knut, Faroogh and I also put down half a day to conduct backlog grooming,

to generate new backlog items for the next sprint.

The team was satisfied with me as Scrum master during sprint 4, so during this sprint I was re-elected to fulfil these obligations. During sprint 5 I worked mainly with Marcus and the implementation of the streaming.

I continued to work on the video streaming with Daniele and Alex developed during sprint 6. After some discussion with Ericsson, we changed some of the streaming, mainly to send the chunks as NetInf messages between the nodes, but without content validation. I also worked with the HTML5 streaming interface. It later turned out that Ericsson also wanted a comparison with the modified chunked data with a pure NetInf streaming solution, I started to implement another HTML5 interface to playback video with pure NetInf while Marcus and Thomas worked on the client backend.

In the beginning of sprint 7 Alex and I finished the HTML5 interface for pure NetInf streaming. Then I conducted an evaluation of the streaming.

A.3 Paolo Boschini

During the first sprint I was in charge of investigating what mobile phone models would be a good fit for running the application we would build during the project. I looked at their specification and chose two models that Ericsson would then provide to us. As a frontend member I then started working on Bluetooth communication between phones together with Kim. We managed to get the phones sending messages to each other and transfer files as this was one important feature our application should support.

In the second sprint I studied a previous implementation of NetInf and tried to understand it in depth to get a valid reference to use during the project. I also wrote the code conventions for our workflow. After that I continued working on the Bluetooth implementation with Kim and implemented programmatic discovering of other devices and the ability to exchange binary information object (BO) between phones.

In the third sprint I took the role of Scrum Master. I got acquainted with Jira, a digital tool for keeping track of stories and sub-tasks. To have a more instant overview about the sprint state I also used post-its on the whiteboard in our office making sure to synchronize them with Jira. I was also responsible for keeping track of the work state of the backend group in

order to facilitate the communication between the two groups. The rest of the time was spent on testing and refactoring. I read up on best practices when following test driven methodology in an Android environment and integrated that into our application. At this point our team felt the need to reorganize and optimize our version control system workflow, so Kim and I reorganized git branches to simplify our version control workflow. The refactoring part consisted of adding utility classes, fixing incomplete code comments and adding license information to our code. I then read up more on Android UI components and refined the UI structure and design of our application. At last I helped Linus to implement functionalities for fetching and posting data to the NRS cache implemented by the backend group.

In sprint four I helped Kim to separate the main application and the NetInf functionalities into two different projects. This decoupling was very important since it makes it possible for other developers to develop their own application and use our existing Netinf Android implementation. Another important feature was implemented in this iteration, namely the routing of network requests to our Netinf service when downloading web resources before displaying them into our application. Since each html page contains resources (images, text, videos) we could save them individually and pass them to our NetInf service as NDO.

The last sprints involved minor bug fixes so that our application would support and correctly display a major number of web pages.

A.4 Kiril Goguev

During the first sprint I helped setup the build tool environment (investigating Buildbot with Alex, Faroogh and Jon). Then, I read OTP, attended the Erlang workshop and started to get a grasp of Erlang.

In the second sprint, I setup Jenkins with Alex and connected the GIT repositories, I also created a document on how to set it up from scratch. Daniele and I designed and implemented the HTTP Handler and the NRS logging service. We also created the GIT practices document and taught the groups how to use the proper workflow for our project. Finally, I along with Thomas helped design and implement the foundation of the NetInf protocol (The internal representation of messages in the system).

In the third sprint Faroogh and I migrated all the data and build tools from the old server to a new server(due to errors). Alex and I fixed merging the metadata in the list database, creating and implementing the plug and play (PNP) database architecture, abstracting the list database to use the new PNP database architecture and finally the content validation. Later, I worked with Marcus to implement content storage and content handler into our system as well as fixing the integration test which was broken when we added all the new modules. Finally towards the end of the sprint, Faroogh and I designed and implemented a UDP discovery protocol to be able to find other NRS' on the network. At the very end of the sprint I started looking into Python SAIL implementation of NetInf but had to abandon it since there was too many problems to fix in order to be able to communicate(this was a problem of the differences in the drafts each implementation followed). I also wrote a wiki article for the plug and play database architecture.

This is the sprint where I felt that I had a good grasp of Erlang and how to code in the proper OTP way.

In sprint 4, I had a part in designing the very first NetInf video streaming draft along with Faroogh and Knut. Later Faroogh and I added truncation to the NRS system and verified that we met all the required components of the netinf protocol draft provided at the beginning of the course by the client.

In sprint 5, I designed a setup/install script. I also designed and implemented the UDP convergence layer from the draft with Thomas and deprecated the UDP discovery protocol Faroogh and I coded in sprint 3. I also took over Faroogh's old task of making the list database persistent. Finally, I wrote the wiki article on how to use the script and test the UDP convergence layer.

In sprint 6, Alex and I organized the Course and Product Reports into separate files. Thomas and I made the system configurable using external files (.config files) that can be loaded on the command line at runtime and I wrote the wiki article for how to write for the reports using the structure we created as well as how to use the configuration files in the system. Also

populated draft sections of the course and product report using all the wiki articles the backend team had written during the course with Daniele. I fixed and finalized the setup/install script.

Finally in sprint 7, I reorganized the structure for the reports into folders and showed people how to use the structure. I wrote parts of the product and course report.

A.5 Faroogh Hassan

During the first sprint the whole back-end group concentrated on learning Erlang and Open Telecom Platform (OTP). A two day workshop on Erlang by Henry Nyström (Campanja) and another workshop by Gustaf Naeser (Hansoft) on Agile software development were also part of this sprint. I worked with Jon to establish the coding standards which we followed throughout the rest of the project. Jon and I also investigated if there are any Extreme Programming (XP) practices that we can incorporate in our development methodology and we came up with a set of recommendations.

I started the second sprint by taking part in developing the overall design of our application. I also worked with Thomas to write code and some tests for message handler and message parser.

Third sprint was the longest sprint of this project (3 weeks) and I got the opportunity to be the Scrum master of the back-end team for this sprint. As Scrum master my major task was to remove any impediments that may arise during the course of development. I worked with Alex on writing for the search functionality. I also worked with Kiril on the NRS discovery protocol where we used UDP messages for the discovery purpose. Apart from that I also wrote unit tests for the modules where unit tests were missing.

In the fourth sprint I was involved in designing the architecture of the streaming video functionality. We wrote a draft specification for this functionality based on the proposed architecture. I also worked with Kiril on

hash truncation.

In the fifth sprint I investigated if our implementation of Netinf protocol is compatible with other implementations. We aborted this task later on the customer's request.

Sixth sprint was the last sprint before the Christmas break where we made some final adjustments to our application and cleaned up our code. I wrote 'specs' and comments in the modules where they were missing.

Seventh sprint was the last sprint of our project and we dedicated the whole sprint to write course and product reports.

A.6 Marcus Ihlar

During the first sprint I studied OTP design principles, investigated existing HTTP libraries for Erlang and implemented a simple demo server together with Thomas. We decided to use Cowboy for HTTP handling. The server partially implemented NetInf publish and get functionality.

The second sprint signaled the start of real product development. In the beginning of the sprint I did an overall design of the intended system with Faroogh and Thomas. When actual coding started I wrote a lot of the boilerplate code necessary to setup an OTP application, later I focused on event-handling logic and test code.

In sprint 3 I implemented content storage together with Kiril, after that we focused on getting the integration test working properly. I designed the architecture for forwarding of NetInf messages together with Thomas and Alex. I implemented message id storage (as part of the distribution architecture) together with Thomas and then updated the event handler to handle message forwarding, this led to alot of re-factoring throughout the system, especially to make message passing asynchronous.

Sprint 4 was short, only one week. I focused on re-factoring and code cleanup, especially in the http message formatting module. We also did some backlog grooming and at the end of the sprint I helped finish the

streaming draft.

During the fifth sprint I implemented streaming functionality together with Jon in accordance to the draft written the week before.

Sprint 6 was my turn to try the role of Scrum master. Being Scrum master at this point felt very straight forward since we were so far into the project and had a good working environment. I did a lot of bug fixing and work on the integration test. Thomas and I rewrote some of the streaming code and started implementing pure NetInf streaming. We ended the sprint with a beer and Quake 3 session!

I started sprint 7 by finishing the pure NetInf streaming in order to be able to run evaluation tests. I also worked on presentation, evaluation and reports.

A.7 Alexander Lindholm

The first few weeks of the project were all about team-building as well as reading up on the subjects ICN and NetInf. The first sprint was mainly about refreshing our functional programming skills as well as setting up all the tools such as Git, Redmine, Jenkins, Emacs etc. We also came to certain agreements such as trying to use test-driven-development.

The second sprint I was involved in the creation of a external protocol draft as well as setting up Jenkins along with Kiril and creating Curl scripts for testing of our system along with Jon. Jon and I also created a first naive version of a storage. At the end of sprint most of us were busy fixing bugs before the presentation at Ericsson in Kista. Here I also spent time on implementing a "beautiful logging system" for the presentation in Kista.

Sprint three started off with the presentation in Kista and it went well. Everyone was pleased with how the presentation went. During the rest of the sprint I mostly worked with Kiril and we were involved in writing a module for validation of hashed content as well as writing code for merging of NDO's in the database as well as fixing the tests for the storage module. Kiril and I also created the plug-n-play database architecture. Other than that Faroogh and I implemented searching of NDO's within our system.

Sprint four I developed modules for forwarding of messages on the HTTP convergence layer. A few days after the sprint had started the backend team started to diverge from the frontend due to several facts; we were

done with the basic NRS functionality in the backend and we were planning on implementing streaming within NetInf, which was something that the frontend didn't have time to do. Therefore I mainly worked with Danielle and Jon this sprint and implemented an HTTP-client that would come to be used mainly for streaming, but also worked as a fully functional NetInf-client.

During sprint five I mainly worked with bug fixing within the system as well as making the HTTP-client more robust and fully functional.

During sprint six we had feature-freezed and my work mainly consisted of fixing bugs as well as refactoring code. A lot of time was spent on fixing a bug that resolved around multipart-Http-data being corrupt after transmission, it turned out that the bug was within Cowboy, the open-source Http-client we used. Kiril and I also created the initial structure for the reports.

In the beginning of sprint seven I worked with Jon on the implementation of a different version of NetInf-streaming that uses pure NetInf for all transmissions instead of a mix of NetInf-messages along with a content-handler which were used in the previous streaming version.

A.8 Knut Lorenzen

In the pre-Scrum phase I made two project proposals: One was to integrate NetInf into the Android OS, the other one to start a new NetInf branch in Erlang. As Linux kernel level development was considered tedious the first idea was dropped immediately. The Erlang idea initially received a luke warm response as it did not sound very original compared to other proposals. However our Ericsson contacts mentioned that they would love to see a NetInf implementation in Erlang as it is "their" own language, and after a few days more and more group members changed their mind, perhaps because they realised it would be a great opportunity to practise and learn Erlang on a real project.

I became the Scrum master of the backend group for the first two sprints since nobody else volunteered. Having worked as a software developer for a few years after my graduate studies and therefore being more experienced, I felt that I would be more useful in overseeing the development process rather than writing code. During that period I spent my time curating the backlog, reviewing people's tasks, coaching them in working test-driven and

setting up and using the development infrastructure. I did not write a single line of code until the third sprint. I also presented the backend group's work during first review at Ericsson after sprint two.

Unfortunately for me (I had really enjoyed being the Scrum master), Olle, the course teacher, requested both teams to appoint new Scrum masters after that, so that others could have the opportunity to practice that role. In the three weeks of sprint 3, my work focussed around creating an integration test. So far, only module tests existed, and the HTTP interface was tested interactively using Curl. I tried hard to convince people to write tests for the (automated) integration test rather than interactive Curl-tests or module tests. I continued to add test cases to the integration test and improve it until the end of the project

In sprint four, I decided to become an additional spokesperson for the backend team. I had the impression that there was a lack of communication between the product owners and us during the previous sprint, i.e. none at all until the demo. This had led to some members implementing features not requested by the product owner. At that point, we had mostly finished implementing the NetInf protocol draft, and the focus of the project changed towards adding streaming functionality. For this, no specification or prior implementation existed. I tried to develop a design draft together with Börje, our contact a Ericsson, through email communication. This did not work out very well, and so we decided to come up with our own design and present it at the demo after sprint five.

A.9 Harold Martínez

In the first sprint, I worked on defining the code conventions that we wanted to use. We discussed them, refined them and I set up the Eclipse plug-in Checkstyle with the chosen conventions. Also, I was selected as the groups' spokesperson for the whole project, so I managed the communication between the customer and the group.

In the following sprint, Kim-Anh, Paolo, Linus and I worked on the first architecture draft. I also worked with Linus on the development of the earliest version of the Resolution Service, which connects with the backend's Name Resolution Service, implementing the *GET* and *PUBLISH* methods.

In the third sprint, I prepared, together with Knut and Thomas, the presentation for Ericsson. I also presented this. Then I worked together with Kim-Anh designing the NDO database that will be used for the Local Resolution Service.

In the fourth sprint, Linus and I created the GUI settings for the Android application, letting the user change some configurations values. I also set up a GitHub account to share the evolution of our project. In order to improve the user experience in the application, I separated the Bluetooth discovery process and created a scheduled discovery that will be triggered from the moment the application starts, running every five minutes.

In the fifth sprint, I began defining and implementing the Bluetooth Convergence Layer, however, due to time constraints and other priorities, this job was not finished.

I was Scrum Master for the whole group for the last sprint when we wrote these reports. I was also in charge of presenting our results at the end of the course.

A.10 Thiago Costa Porto

When the project started, I read a few papers about ICN and tried to focus on the issues that we were going to face going forward. We had meetings to decide what our project would be like and I felt I was very active in those days. One thing that helped was my previous experience with Scrum, which lead to greater understanding of it by doing it at the university level, giving you the chance to try things the proper way, so to say. When time came, I chose the frontend team because I thought it would be fun to work on the "client", shaping it to the way that we had planned.

I was the Scrum master for the first two sprints. In the first sprint, I set up Jira, our project tracker tool, and did all the things the Scrum master should do. I was very focused on getting Scrum to work, at the same time I coded

small features for our client. I focused on the networking side of the client, and spent several hours understanding how Android works and how the code we had at our disposal worked. In the second sprint, I could focus more on developing the application and less on Scrum – everything was going as it was supposed to go – and I wrote a few of the classes used on the client's backend. I also provided support for measurements (download/upload) and added early support for metadata during transfers.

On the third and fourth sprints I focused mainly on getting the search functionality to work properly. In the third sprint, I provided a solution that was out of the "NetInf" architecture, and I focused on integrating it to NetInf using the Resolution Services on the fourth sprint, together with Linus. Apart from the search, I also refactored some code and helped document our code, helped a little with the design of the database, started user feedback and setting up our revised workflow. In the fourth sprint, I worked a lot with JSON, communication with the NRS and making responses uniform throughout our application, with Linus.

Near Christmas, I defined the Evaluation with the other team members and started working on that. I implemented the logging functionality together with Linus and did some extra refactoring on the side. Close to the finish line, I spent time documenting the code for further usage.

This was a very interesting course because it not only simulates a work environment, but provides a lot of insights on your own behavior and on team management. It is very good working with people from different places and with different backgrounds. I definitely recommend this course.

A.11 Linus Sunde

During the first sprint I attended the Erlang Workshop which was mainly directed towards the backend group. This made it natural for me to work with integration between the groups. Together with Thomas and Marcus I worked on sending NetInf messages using HTTP between our application and their server.

In the second sprint I sat down with Jon and Alex and discussed the protocol draft for the NetInf HTTP convergence layer. This discussion resulted in an initial specification. The frontend group decided to use OpenNetInf in our application. I spent some time incorporating OpenNetInf into our project.

After this I worked with Harold to create the Resolution Service which communicates with the backend's NRS. This also involved working together with Thomas and Marcus from the other group to solve integration issues as they popped up. Integration issues kept popping up during all the coming sprints and I spent a huge amount of time working on these, most often together with someone from the other group.

I started sprint three with creating a paper prototype together with Kim and Paolo, in preparation for the first review in Kista. The review went well and I felt the paper prototype really gave us a more concrete feeling of our goal. After the review I looked into speeding up the compile time of our application as it was making testing of small changes slow and painful. I also spent a lot of time refactoring, which in hindsight was time well spent. As for new functionality, I worked together with Paolo creating some setup dialogs and downloading of simple web pages using NetInf, as well as sending and receiving cached files from the backend's NRS. Finally I read up on testing using Android JUnit and created tests for some of my code.

During sprint four I once again spent time refactoring, and once again I feel it was time well spent. I worked with Harold to create a settings menu for our application and to make the program use these settings. I worked with Paolo to create a way for our application to know if files were acquired using the Internet, Bluetooth, the NRS cache, or the database on the phone.

I was Scrum Master during sprint five. I spent some time making sure some backend fixes we needed were implemented by the backend. Other than that I mainly worked on preparing evaluation together with Thiago and Paolo. We implemented logging functionality to be able to gather some statistics for our report.

The last sprint was dedicated to writing the reports.

A.12 Kim-Anh Tran

In the first sprint I read up on JIRA and presented the ways to use JIRA within our project. Afterwards I mainly worked together with Paolo on establishing the first Bluetooth communication, so that we could handle file requests from other connected devices.

During the second sprint Harold, Paolo, Linus and I created the first draft

of our architecture using OpenNetInf. Thereafter Paolo and I continued on our previous Bluetooth implementation. We added the Bluetooth discovery and more importantly the functionality to request and transfer NetInf NDO between devices.

The third sprint I worked on reconfiguring Jenkins with Git and writing a document on our git workflow. I added a test project for our code that was run by Jenkins. Afterwards Harold and I worked together in order to develop a database for storing NDO and thus a Local Resolution Service.

Sprint four involved separating our current application: one that provides only the NetInf services and one that contains our application which uses these services. With help from Paolo I separated the two projects and resolved all dependencies. Afterwards I joined Paolo in order to finish parsing HTML files to be properly displayed within our WebView component while using our NetInf services. During this sprint I was Scrum Master. Amongst other tasks I needed to attend the other group's stand-up meeting, create the stories and tasks in JIRA and update our Scrum board. It was a good experience to take the role as a Scrum Master and to organize our Sprint.

Within the last sprint I solved the text encoding problem when displaying a number of web pages. I cleaned up our logs so that they were readable and more useful during debugging. Finally I created a JAR file for libraries that were used in both applications.

A.13 Thomas Nordström

Pre-Scrum: In the first few weeks the entire Project CS team read up on ICN in general and NetInf in particular while trying to decide on what our project would be. We also had team-building exercises.

Sprint 1: We all set up our working environment and had two workshops, one on Erlang and one on Agile development. I also read up on OTP, did some example programs, helped other team members get a hang of functional programming and worked a little on the simple demo server with Marcus.

Sprint 2: I worked on the message parsing and handling with Faroogh. I also worked on the first draft of the internal message specification and implementation with Kiril. At the end of the sprint I worked with Linus in the front end to make our systems work with each other.

Sprint 3: First I did some re-factoring of the code, after that I worked on the search both in the message handling and in the list database. I also worked on the presentation at Ericsson with Knut.

Sprint 4: This sprint I fixed some minor bugs, added some statistics keeping in the server and worked on the backlog grooming.

Sprint 5: I started working on UDP convergence layer with Kiril.

Sprint 6: I started the sprint fixing a lot of minor things and while doing that I found a bug in one of our open source dependencies and got that fixed.

Sprint 7: In this sprint the entire group worked on report writing and the final presentation.

Appendix B

Git Workflow

The following section will detail a sample work flow for a sprint with at least two story items.

The figure below B.1 shows how the individual story may look on a time-line.

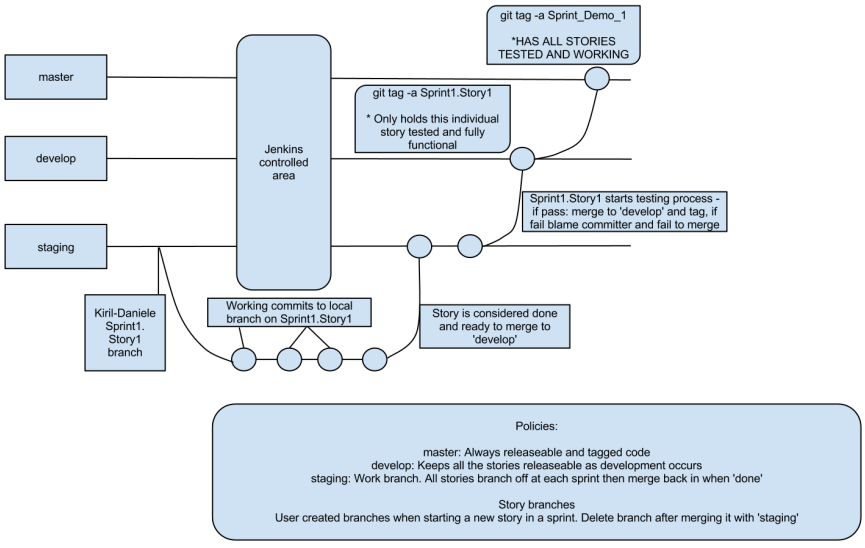

Figure B.1: Sample workflow for one story item

B.0.1 Starting a sprint

At the start of each sprint the development team will perform a git checkout -b SprintX.shortStoryName develop for a running example we will take the HTTP Handler from the first sprint. This guide assumes that all coding will be done in pairs.

```
git checkout -b SprintX.shortStoryName develop
git push -u origin SprintX.shortStoryName
```

This will pull all the previous items into the temporary working branch from develop

B.0.2 Working during a sprint

As the developers work together to complete the tasks outlined in the story, commit as usual to the local branch.

```
git pull origin staging
git add [files that have been added or created/modified]
git commit -m description - where description is the commit
message you would like associated with the commit.
```

B.0.3 Deploying a Done story

After all the tasks are completed and the pair feels ready to move the task to the done state, merge it to the development branch using the following command:

```
git checkout staging
git merge --no-ff SprintX.shortStoryName
git push origin staging


- do the following only if you wish to
```

```
delete your branch on the remote repository -
git push origin :SprintX.shortStoryName

- do the following if you wish to delete your branch locally -
git branch -d SprintX.shortStoryName
```

This will perform a merge and fast-forward on the remote 'staging' branch while keeping historical information. It will also create empty commits in the development branch as a way to quickly spot revisions which can be used to revert the code. For example if you need to remove a story that was not working properly.

NOTE: As a human user this is typically where you stop working in the workflow. The following steps are to be implemented by Jenkins and automated build tools. A developer should only be concerned with working on the staging branch and your their own story branches for each sprint. Never touch develop, master or release unless absolutely necessary.

B.0.4 Testing a sprint story

The following steps assume that a testing suite was created and works well with the Jenkins build tool.

B.0.5 Merging in to develop

When a story branch has been merged to 'staging', the code goes through a set of unit tests and when they are successfully passed the code is ready to be merged to 'develop':

```
git checkout develop
git merge --no-ff staging
git push origin develop
```

B.0.6 Merging in to release

When a story branch has been successfully merged to 'develop', the code goes through a integration test and if it successfully passes it is ready to be

merged to 'release':

```
git checkout release
git merge --no-ff develop
git tag -a SprintX.shortStoryName-RELEASE
git push origin release
```

B.0.7 Merging in to master

Supposing that all the sprint stories code have reached the stage of release, before proceeding with the final merging, the developers check that all the desired features have been implemented and tested properly. The final merging is performed upon the master branch which always contains the stable version of all the code written so far.

```
git checkout master
git merge --no-ff release
git tag -a Sprint1.Demo1
git push origin master
```

B.0.8 Post-sprint practice

After each sprint the build tool should synchronize the 'master' and the 'develop' branch. This ensures a clean start for each sprint as the 'staging' branch will inherently be considered the "dirty" branch.

```
git checkout DEVELOP
git merge --no-ff MASTER
```

B.0.9 References

The following was an inspiration for the git model:

http://nvie.com/posts/a-successful-git-branching-model/

Appendix C

Jenkins Setup

The following guide describes exactly how the ERNI team installed and configured the automated build tool for use during the course.

C.1 Building Jenkins

Make sure you download the latest *.deb file from the Jenkins website:

http://pkg.jenkins-ci.org/debian/

To install simply run the following command in a shell:

run Dpkg -; <Path_to_deb_file>

After installed, Jenkins by default starts as a service and runs on port 8080.

C.2 Jenkins configuration

Clicking on Configure system from the Manage Jenkins button will allow you to configure the overall settings of Jenkins. Please note you can also

configure *SOME* plugins from here as well. If you do not see the plugin configuration you have installed here, then take a look in the jobs configuration to find settings - if any.

C.2.1 Java JDK

Before configuring the Java JDK please install it on the server you are working on. It is required for the Email notification service and also required for Android project building.

1. Name: <enter any name>
2. JAVA_HOME: <enter the exact directory on the server machine that holds the bin for jdk >

C.2.2 Android tools

Before configuring this in Jenkins makes sure you have installed the Android SDK in the server you are working on.

1. Enter the path on the server for the Android SDK Root - In our case it is: /home/server-build/android-sdk-linux
2. Check off Automatically install Android components when required

C.2.3 Ant

1. Enter a Name for the Java ANT environment on the server
2. Check off install automatically.

C.2.4 GIT

Our version of Jenkins requires GIT plugin which can be installed from the manage plugins page. Scroll down and look for the Jenkins GIT plugin or use the filter in the top right hand side of the page

Setting up the GIT plugin is done from the overall settings page under the section Git plugin.

Enter the following settings:

1. Global Config user.name Value: <enter a user name for the git pusher>
2. Global Config user.email Value: <enter the email for the git pusher>
3. Create new accounts base on author/committer's email : checked - this will automatically create people in the Jenkins people page with thier username and email. Also a lookup for the email notifications.

C.2.5 JIRA

Install the JIRA plugin from the manage plugins page. Look for Jenkins JIRA plugin.
Setting up the JIRA plugin is done from the overall settings page under the section JIRA.

1. URL:<enter the url for the JIRA server>
2. Supports Wiki notation: checked
3. Update Jira for All Build Issues: checked
4. User Name: <enter the name for the JIRA user bot which will make the updates on JIRA as builds occur
5. Password:<enter the password for the JIRA bot>

C.2.6 Email server

If you want to have emails being sent from Jenkins the best way is to use the google SMTP server. You will want to set up a project email address at google mail.

Enter the following settings in the Email Notification section:

1. SMTP server: smtp.gmail.com
2. Sender Email-Address: project.cs.2012.uu@gmail.com
3. make sure USE SMTP Authentication is checked.

Click advanced

1. Enter the email address name when you set up the account- in our case it is: project.cs.2012.uu@gmail.com
2. Enter the password for the account
3. Check SSL
4. SMTP port set to 465
5. Char-Set is at default: UTF-8

C.2.7 Projects

Jenkins requires Projects in order to do anything useful.

To create a new job make sure you are on the jenkins tab, in the top left corner of the webpage. Select **New Job**

On the screen

1. Enter a Job name - This name will show up on the main dashboard page
2. Select Build a Free-Style software Project.

Click OK when finished and you will be taken to the Configure screen of the project. Our specific project has the following project properties

(NOTE: steps 14-17 is for erlang only)

1. Description - enter a project description

2. under Source Code Management Select Git
3. In Repository URL - enter the url for the git repository you will pull from(in our case /home/git/repositories/backend.git)
4. In Branches to build -enter the exact branch you wish to pull and build from. in our case it is origin/BRANCH NAME i.e origin/STAGING origin/DEVELOP , origin/RELEASE, origin/master.
5. under Build Triggers Select Poll SCM - enter the time to have jenkins check the git i.e * * * * * for every minute.
6. under Build click add Build Step and select Execute Shell
7. in the Execute Shell command box - enter any commands required on the commandline in order to build your project. i.e cd netinf_nrs ; make etc...
8. under Post-Build Actions click Add post-build action and select Build other projects
9. In Projects to Build - enter the name of the project you wish to start after the selected trigger has been fired.
10. select the trigger Trigger only if build succeeds(used to make sure nothing went wrong with the current project steps).
11. under Post-Build Actions click Add post-build action and select Email Notification - this action uses the email list built in the people directory in the jenkins main dashboard.
12. check Send e-mail for every unstable build
13. check Send seperate e-mails to individuals who broke the build
14. under Post-Build Actions click Add post-build action and select Publish Cobertura Coverage Report - this will pull erlang eunit and coverage tests into the project main dashboard.
15. In Cobertura xml report pattern - enter **/*.coverage.xml
16. under Post-Build Actions click Add post-build action and select Publish Junit test Report
17. In Test report XMLs - enter **/.eunit/*.xml

C.3 our GIT workflow with Jenkins

The backend team will use the workflow outlined in the report section Git Workflow. As such the following build projects will be configured for use with Jenkins

C.3.1 Backend_Staging

1. Description - "This Job will pull from the Backend Git Repoisitory branch: staging and only check if the source files compile. If they compile then they are passed on to the develop branch."
2. Git Section - URL /home/git/repositories/backend.git
3. Git Section - Advanced - Name: staging
4. Git Section - Branches to build: staging
5. Build Triggers-Poll SCM - * * * * *
6. Build -Execute Shell - cd netinf_nrs; make compile
7. Post Build Actions - Git Publisher Push only if build succeeds -checked
8. Post Build Actions - Git Publisher Branch to Push: develop
9. Post Build Actions - Git Publisher Target Remote Name: staging
10. Post-Build Actions - Build other projects: Backend_Develop

C.3.2 Backend_Develop

This job assumes all code already compiles when it enters this branch. This branch is specifically for unit tests.

1. Description -"This job will pull from the Backend Git Repository branch: develop and checks if the source files compile and pass the unit tests. If they compile and pass the tests then they are passed on to the release branch."
2. Git Section - URL /home/git/repositories/backend.git

3. Git Section - Advanced - Name: develop
4. Git Section - Branches to build: develop
5. Build -Execute Shell - cd netinf_nrs; make compile; make eunit;
6. Post Build Actions - Git Publisher Push only if build succeeds -checked
7. Post Build Actions - Git Publisher Branch to Push: release
8. Post Build Actions - Git Publisher Target Remote Name: develop
9. Post-Build Actions - Build other projects: Backend_Release

C.3.3 Backend_Release

This job assumes all code already compiles AND passes the unit tests. This branch is specifically for integration testing.

1. Description -"This job will pull from the Backend Git Repository branch: release and checks if the source files compile, pass unit tests and finally pass integration tests. If they compile, and pass all the tests then they are passed on to the master branch(Demo)."
2. Git Section - URL /home/git/repositories/backend.git
3. Git Section - Advanced - Name: release
4. Git Section - Branches to build: release
5. Build -Execute Shell - cd netinf_nrs; make compile; make eunit; make integration

If the final Backend_Release succeeds then the following job Backend_Demo should be taken care of manually until the process is perfected.

C.3.4 Backend_Demo

This job assumes all code already compiles AND passes the unit tests. This branch is specifically for tagging demos.

1. Description -"This job will pull from the Backend Git Repository branch: release and checks if the source files compile, pass unit tests and finally pass integration tests. If they compile, and pass all the tests then they are passed on to the master branch(Demo)."
2. Git Section - URL /home/git/repositories/backend.git
3. Git Section - Advanced - Name: release
4. Git Section - Branches to build: release
5. Build -Execute Shell - cd netinf_nrs; make compile; make eunit; make integration
6. Post Build Actions - Git Publisher Push only if build succeeds -checked
7. Post Build Actions - Git Publisher Add tag - Tag to push: DemoX (where X is the sprint number)
8. Post Build Actions - Git Publisher Target Remote Name: release
9. Post Build Actions - Git Publisher Branch to Push: master

Appendix D

Erlang Coding Standards

The purpose of this section is to define coding standards for the NetInf implementation in Erlang. This document is based on the official coding standard and conventions for Erlang [1].

D.1 Engineering Principles

D.1.1 Export As Few Functions As Possible From a Module

For better readability and understanding of code, it is recommended that we export as few functions as possible. When exporting a function you should define specification, also known as `contract`, with the erlang -spec compiler attribute [2].

D.1.2 Prefer Readability Over Speed

It is recommended that code is initially written in an easy-to-read manner instead of writing code that makes the program run fast but hard to understand. However, if run time complexity becomes a problem, consider rewriting the code in an optimized way.

[1] `http://www.erlang.se/doc/programming_rules.shtml#REF87730`
[2] `http://www.erlang.org/doc/reference_manual/typespec.html#id75681`

D.1.3 Directory Structure of an OTP Application

Erlang/OTP applications should have a directory structure as shown below:

```
<application-name> [-<version>]
                   |
                   |-doc
                   |-ebin
                   |-include
                   |-priv
                   |-src
```

Where:

doc This is where the overview.edoc file goes, if one wants to generate documentation from EDoc.

ebin This is where compiled code (.beam files) and meta data file (.app) are located.

include Public header files (.hrl file) should be kept in this directory.

priv Template files, shared objects and DLLs.

src Application source code, which includes .erl files and internal .hrl files. It can also have ASN.1, YECC, MIB and other source files.

D.2 Specific Lexical and Stylistic Conventions

Use Emacs and Erlang mode when writing code. To correct indentation, just press Tab when you are in Erlang mode. We recommend that Emacs is used for writing the code because everybody in the team will be familiar with everyone else's development environment. This is important in case team members decide to do pair programming.

- Do not write deeply nested code which means not more than 2 levels of indentation.
- Do not write larger modules than 400 lines.

- Do not write long lines, at most 78 characters.
- Do not write longer functions than 15 to 20 lines.
- Choose meaningful variable names and use capitalized letter to separate the words. Example: `ReceivedMessage`
- The function name should represent what the function does. Use underscore to separate different words. Example: `send_message()`.
- Give space after each argument in a data structure or function arguments. Example `12, 13, 45`.
- Use tagged return values. This means that the return value of the functions should be a tuple where the first key should be an atom explaining what the tuple is.
- Do not use -import
- Use multiple -export clauses instead of grouping a lot of functions together in one export clause. Examples: user interface, intermodule exports and exports for use within module only.
- Always use proper indentation.
- Do not comment out old code, just remove it because it is already in the source control repository.

D.2.1 Comments and Documentation

Instead of littering your code with comments, you should write meaningful variable and function names. But in some cases where the code is hard to understand you can write short comments. For every function use Edoc notation and explain arguments and return value and, if present, side effects [3].

Each file should start with a short description of the module contained in the file and a brief description of all exported functions. All the error messages in an application should be documented in a single document.

[3] `http://www.erlang.org/doc/apps/edoc/chapter.html`

Appendix E

Java and Android Coding Standards

The purpose of this section is to present the code convention used in the implementation of the Android-based application. These code standards are based on the Android Code Style Guidelines for Contributors[1].

E.1 Java Language Rules

- Do not ignore exceptions. It is only acceptable to ignore exceptions if there is a good reason to do it, which should be given as a comment in the code. As a general rule, always:
  - handle the exception;
  - or throw a new exception according to the level of abstraction;
  - or handle it gracefully;
  - or throw a new *RuntimeException* in case there is nothing possible to do.
- Don't catch a generic exception e.g. *Exception e*. Alternatives to do this are:
  - Catch each exception with a separate catch block after a single try.

[1] `http://source.android.com/source/code-style.html`

  - Refactor the code in order to have multiple try blocks.
  - Rethrow the exception and let it be handled on the next level.
- Do not use finalizers. Let the garbage collector do its job.
- Always write full imports.

E.2 Java Style Rules

- Use Javadoc standards for commenting code. Always write the descriptions in third person. This rule can be skipped if the method is too trivial e.g. a *getters* and *setters*.
- Try not to exceed 40 (forty) lines of code for each method.
- Try to keep the scope of a variable as small as possible. Also, try to initialize it with a proper value.
- Order the imports beginning with Android libraries, followed by third parties libraries and ending with *java* and *javax* classes, each group separated by an empty line.
- For indentation, use four spaces for blocks and eight spaces for line wraps (i.e. when the line of code is too long and needs to be cut).
- Limit the line length to 80 (eighty) characters.
- Use standard Java Annotations e.g. *@Deprecated, @Override, @SuppressWarnings*.
- Use acronyms as words e.g. write *XmlHttpRequest* not *XMLHTTPRequest*.
- Use TODO comments for temporary fixes and future work.
- Use all five levels of logging (Error, Warning, Informative, Debug and Verbose) as applicable.
- Use the standard brace style:

```
public void foo() {
    if (...) {
        doSomething();
    }
}
```

- Use the following naming conventions:
  - Non-public, non-static field names start with m.
  - Static field names start with s.
  - Other fields start with a lower case letter.
  - Public static final fields (constants) are letters in upper case and spaces replaced by underscores. Example:

```
public class MyClass {
    public static final int SOME_CONSTANT = 42;
    public int publicField;
    private static MyClass sSingleton;
    int mPackagePrivate;
    private int mPrivate;
    protected int mProtected;
}
```

Bibliography

[1] Buildbot - an automated build tool. Retrieved February 4th, 2013, from http://trac.buildbot.net/.

[2] Ericsson - a world of communication - ericsson. Retrieved January 15th, 2013, from http://www.ericsson.com.

[3] Erlang programming language. Retrieved January 15th, 2013, from http://www.erlang.org/.

[4] Gerrit code review. Retrieved February 18th, 2013, from http://code.google.com/p/gerrit.

[5] Git - fast version control. Retrieved February 4th, 2013, from http://git-scm.com/.

[6] Jenkins - an extendable open source continuous integration server. Retrieved February 4th, 2013, from http://jenkins-ci.org/.

[7] Jira - an issue & project tracking software. Retrieved February 4th, 2013, from http://www.atlassian.com/software/jira/overview.

[8] Redmine - a project manager web application. Retrieved February 4th, 2013, from http://www.redmine.org/.

[9] Riak - open source, distributed database. Retrieved February 18th, 2013, from http://basho.com/riak.

[10] B. Raghavan S. Shenker A. Singla J. Wilcox A. Ghodsi, T. Koponen. Information-centric networking: Seeing the forest for the trees. *ACM HotNets-X*, 2011.

[11] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 2012.

[12] P. Boschini K. Goguev F. Hassan M. Ihlar-A. Lindholm K. Lorenzen H. Martínez T. Nordström T. Costa Porto L. Sunde K-A. Tran D. Bacarella, J. Borglund. Project cs 2012 product report. 2012.

[13] H. Otaola and M. Sosa. Using multiple transport networks in netinf enabled android devices. Master's thesis, KTH, School of Information and Communication Technology (ICT), Sweden, 2012.

[14] C. Dannewitz B. Ohlman A. Keranen P. Hallam-Baker S. Farrell, D. Kutscher. Naming things with hashes draft-farrell-decade-ni-10. Handed to us by Ericsson, 2012.

[15] K. Schwaber and J. Sutherland. *The Scrum Guide*, 2011.