Muksis project

Project report

Richard Domander
Tuomas Mäenpää
Teemu Nisu
Tommi Teistelä

Version: 1.0
Public
16th January 2009

University of Jyväskylä

Department of Mathematical Information Technology

Jyväskylä

Approver	Date	Signature	Name verification
Project manager	__.__.2009		
Customer	__.__.2009		
Supervisor	__.__.2009		

Document information

Authors:

- Richard Domander (RD) dimadoma@jyu.fi 050-3482668
- Tuomas Mäenpää (TM) tutamaen@jyu.fi 040-7600465
- Teemu Nisu (TN) tejonisu@jyu.fi 040-8349310
- Tommi Teistelä (TT) totateis@jyu.fi 045-6528709

Document title: Muksis project, Project report
Pages: 34
Source file: project_report-1.0.tex

Abstract: This is the project report document of the software project Muksis. This document describes the follow-through of the project and analyzes how well the project plan was followed during the project. The document begins with explaining the accomplishments and the resources of the project and after that the process and schedule are described. In the later part of the document the management methods and risks are analyzed. In the end the personal experiences of the project members are revealed.
Keywords: Software project, black frame detection, DVB, MPEG, search in MPEG TS, audio filter, MPlayer.

Version history

Version	Date	Changes	Authors
0.1	17.11.2008	The framework of the document.	TM
0.2	24.11.2008	Introduction.	TM
0.3	13.12.2008	Terms, Resources and Process and schedule.	TM
0.4	15.12.2008	Accomplishments, Tasks, workload and division of labour, Management methods, Risks and Personal experiences.	RD, TN, TM, TT
0.5	13.01.2009	Corrections for the project report that were found during the document's inspection.	RD, TN, TM, TT
0.6	15.01.2009	Some more corrections to Management methods and Risks.	TM

Project information

The software project Muksis designed and implemented new features to the open source media player application MPlayer. The customer of the project was Matthieu Weber, who needed these features in his home theatre environment. The implemented features were black frame detection for skipping commercials, support for DVB subtitles, search in MPEG TS to complete the commercial skip and audio filters for compressing and normalizing the audio signal.

Authors:

- Richard Domander (RD) dimadoma@jyu.fi 050-3482668
- Tuomas Mäenpää (TM) tutamaen@jyu.fi 040-7600465
- Teemu Nisu (TN) tejonisu@jyu.fi 040-8349310
- Tommi Teistelä (TT) totateis@jyu.fi 045-6528709

Customer:

- Matthieu Weber mweber@mit.jyu.fi 014-2603056

Supervisors:

- Ville Isomöttönen vilisom@cc.jyu.fi 014-2604976

Contact information:

- Archives: https://korppi.jyu.fi/kotka/servlet/list-archive/muksis/
- Mailing lists: muksis@korppi.jyu.fi
- Room: AgC 225.3 / 014-2604971
- SVN: svn+ssh://svn.cc.jyu.fi/srv/svn/muksis/
- Trac: https://trac.cc.jyu.fi/projects/muksis/wiki
- WWW: http://sovellusprojektit.it.jyu.fi/muksis/

Contents

1 Introduction **1**

2 Terms **2**

3 Accomplishments **4**
3.1 Software 4
3.2 Documentation 5
3.3 Learning goals 6

4 Resources **7**
4.1 Resource constraints 7
4.2 Physical resources 7
4.3 Training 8
4.4 Supervision 8

5 Process and schedule **9**
5.1 Process model 9
5.2 Schedule 9
5.2.1 Iteration schedules 9
5.2.2 Iteration steps 10
5.3 Weekly working hours 10
5.3.1 Richard Domander 11
5.3.2 Teemu Nisu 12
5.3.3 Tommi Teistelä 13
5.3.4 Tuomas Mäenpää 14

6 Tasks, workload and division of labour **17**
6.1 Division of labour by phases 17
6.1.1 Richard Domander 18
6.1.2 Teemu Nisu 19
6.1.3 Tommi Teistelä 20
6.1.4 Tuomas Mäenpää 21

7 Management methods **22**
7.1 Meetings 22
7.2 Communication 22

7.3 Task management . . . 23
7.4 Time logging . . . 23
7.5 Responsibilities . . . 24
7.6 Documentation . . . 24
7.7 Version management . . . 24

8 Risks **26**
8.1 Client's availability . . . 26
8.2 Supervision . . . 27
8.3 Inability to participate . . . 27
8.4 Human relationships . . . 27
8.5 Inexperience . . . 27
8.6 Programming skills . . . 28
8.7 Scheduling . . . 28
8.8 Communication . . . 28
8.9 Language . . . 28
8.10 MPlayer development . . . 28
8.11 Working environment . . . 29
8.12 Complexity . . . 29

9 Personal experiences **30**
9.1 Richard Domander . . . 30
9.2 Teemu Nisu . . . 31
9.3 Tommi Teistelä . . . 31
9.4 Tuomas Mäenpää . . . 32

10 Bibliography **34**

1 Introduction

Muksis was a student software project at the Department of Mathematical Information Technology, University of Jyväskylä. During the fall of 2008 the project designed and implemented new features to the open source media player application MPlayer [1]. The new features were black frame detection for skipping commercials in recorded video file, support for DVB subtitles, seek function to skip commercials based on black frame locations, and audio filter for compressing and normalizing the audio signal. The project team decided to improve similar existing features found in patches written by project client. The software was made for Matthieu Weber who is a senior assistant in the Department of Mathematical Information Technology at the University of Jyväskylä.

The project was performed in a team of four students: Richard Domander, Tuomas Mäenpää, Teemu Nisu and Tommi Teistelä. Tuomas Mäenpää acted as the project manager. Ville Isomöttönen worked as a supervisor for the project. There was no technical supervisor in this project.

The project was concluded in six iterations. The first iteration was called the 0-iteration and began in 11.9.2008. Rest of the remaining iterations completed approximately in two weeks.

This is the project report document of the software project Muksis. The document describes the progress of the project and analyzes how well the project plan was followed during the project. It focuses on the main objectives of the project and explains how well these were accomplished. The document begins with explaining the main terms in Chapter 2. Accomplishments of the project are listed in Chapter 3 and the resources used during the project are described in Chapter 4. After these the process and schedule are described in Chapter 5 and then the tasks, workload and the division of labour are explained in Chapter 6. In the second half of the document the management methods are described in Chapter 7and risks are analyzed in Chapter 8. Finally the personal experiences of the project members are revealed in Chapter 9.

2 Terms

Following terms appear in this document:

Agile software development	a software development process. Agile methods emphasize real-time communication and working software as the primary measure of progress using iterative development.
Software project	course at The Department of Mathematical Information Technology.
C	a general-purpose programming language.
Demuxing	means the same as demultiplexing, the opposite of multiplexing.
DVB	Digital Video Broadcasting, a set of open standards for digital television. Defines various details about the physical and data link layer-level transmission of data, refers to existing MPEG standards for the actual format specifications where possible. The data stream itself is an MPEG-2 Transport Stream with some DVB-specific constraints and may contain multiple channels.
IDE	integrated development environment is an application that provides tools for software development.
Iterative development	technique of developing and delivering incremental components of business functionality. A single iteration results in one or more bite-sized but complete packages of project work that can perform some tangible business function. Multiple iterations recurse to create a fully integrated product.
MPEG	The Motion Picture Experts Group, a working group of ISO/IEC. Also a common name for certain standards created by them.

MPEG-2	The MPEG standard specifying video, audio and related format specifications, primarily used for DVDs and digital television broadcasting.
MPEG TS	is a MPEG transport stream. It is a communication protocol for audio, video and data. It's goal is to allow multiplexing of digital video and audio and to synchronize the output.
Multiplexing	a process where multiple digital streams are combined into one stream over a shared medium.
patch	is a small piece of software designed to fix problems with or update a computer program. This includes fixing bugs, replacing or adding features and improving the usability or performance.
OSS	open source software, a computer software, which source code is made available under a copyright license or arrangement. This permits users to use, change, and improve the software, and to redistribute it in modified or unmodified form.
Subversion	is a free version control system. It is used to maintain current and historical versions of files such as source code.
SVN	is an abbreviation for Subversion.

3 Accomplishments

This chapter analyzes the accomplishments the project Muksis achieved during the project.

3.1 Software

After the team finished the initial research of the MPlayer codebase, working prototypes of the planned features were generally finished quickly thanks to the documentation, old code and other advice provided by Matthieu Weber. The lack of internal documentation, comments and proper specification of features in MPlayer's code (along with the team's inexperience with it) kept providing plenty of surprises until the end of the project, though.

The black frame detection feature was the first one completed as it was implemented as a video filter, separated from most other code, and the interfaces it depended on were virtually unchanged between versions. The team had also looked at using the existing black frame detection filter from MPlayer's newer versions, but it was intended for a different usage scenario involving external tools, and it proved easier to simply port the feature from Weber's old code. The resulting implementation was found to behave in the same way as the old one.

The subtitling and seeking features proved more tricky as both involved considerable interaction with existing code that was difficult to understand due to reliance on "magic numbers" and other code wizardry. The major refactoring the MPlayer code has been going through on its way towards a "1.0" release added a layer of difficulty to comparing the source code of different versions to isolate changes in behaviour. MPlayer on its own never had working support for the subtitling formats commonly featured with MPEG Transport Streams, and it seems that the MPEG TS demuxer's support for subtitles had been subtly broken (at least for our purposes) in some version preceding our target without anyone noticing, much to the team's confusion. Both features were eventually gotten to work.

One additional feature that was thought up during a meeting was audio filtering suitable for a "quiet listening" scenario in a home theatre environment. This was successfully implemented by replicating a typical DVD player's dynamic compression filter and chaining it together with MPlayer's own audio normalization filter-

ing feature, delivering a fairly constant volume level free of sudden peaks that might prove unpopular with family members.

3.2 Documentation

Besides the software itself, the project produced a number of documents related to its development. The documents were written in English, except the ones created for the peripheral course, which were in Finnish. The documents were collected into two folders: One for reference in the university's future software projects and one for the customer. The software patches and the documentation were also burned onto CD-Rs: Two for the project folders, one for each project member and one for the department's archive.

The documents that were completed during the project (in chronological order) are:

- **GPL agreement** contract on licensing the source code and the software under General Public License.
- **Project plan**, which introduced the project, its background and goals, and described its overall management; the schedule, the tasks, how they were divided and what risks the project faced.
- **Application report**, which explained the overall structure of the MPlayer software.
- **Test plan & report** shortly outlined the testing strategy, the test cases, and reported on the test results.
- **Project presentation reports** (esittelyraportit) for the peripheral course.
- **Project report** is this document and was written at the end of the project to analyze and summarize the project progress.
- **Meeting agendas, minutes** and related materials.
- **Source code** a print out of the code that the project team wrote.
- **Project's e-mail list archive** in printed form.

The need for this much documentation was questioned occasionally by the project members, but all project goals that were set for the documentation were met.

3.3 Learning goals

In general, the team members were satisfied with how their learning goals mentioned in the project plan were met. Plenty of other things besides those were learned too as working on a project like this as a team was a first for most of the team – everyone became quite familiar with LaTeX and its quirks, too. Tuomas would have liked to learn more C language, but was happy with the experience he got from being a project manager.

4 Resources

This chapter describes the resources and how the team utilized them. The training and supervision that were arranged by the Department of MIT are also described here.

4.1 Resource constraints

For completing the software project course, the students are expected to work between 270 and 400 hours for the project (10-15 credits for the course) [2]. The team completed the project with in approximately 1180 man-hours. This grants the students 10 credits each. The team members didn't have many other courses simultaneously, so finding time for the project wasn't really a problem. Sometimes though it seemed that preparing for an exam took so much time that some tasks were not completed as planned.

4.2 Physical resources

The project team used the room AgC225.3 inside the project premises AgC223.1. The room was equipped with four computers (one for each member), office supplies and white boards. The computers were generally found fast enough and useful. Since installing programs was prohibited, the team requested installations from MIT's PC support. The team chose to request Linux for all the computers, but it was soon discovered that the Excel document that were used to log working time didn't work under OpenOffice.org and one computer was changed back to work as a Windows machine. The team chose to use the Code::Blocks IDE as their main tool for programming, but compiling MPlayer with it was difficult and eventually it was only used as a favoured editor. The main documents were written with Texmaker, a free cross-platform LaTeX editor and with Microsoft Excel.

The printer in the project premises was generally used weekly and found essential, but the copier of the department of MIT was used only a few times. A small library of technical books was available, but since most of the technical documents were provided by the client, the library wasn't needed. The other results found from the project folders in the project premises were very useful, especially during the

beginning of the project. The team could also get a projector and a digital audio recorder for the meetings. The projector was used in every meeting, and it was useful during the first presentation for demonstrating the implemented features, but the team didn't see any need for the audio recorder. The project premises could also be reserved for the meetings and every official project meeting was held in there.

4.3 Training

During the project the team members attended a peripheral course and got training for various subjects:

- Project leading and management
- Revision Control
- Usability
- Copyrights
- Presentations

The lectures were generally found interesting, though some of the sessions felt quite long, which made it difficult to concentrate on them. The team found the project leading and revision control lectures very useful, but there was little use for the information given in the usability lectures as MPlayer's usability was hardly the focus of the project.

4.4 Supervision

Ville Isomöttönen from the Department of the Mathematical Information Technology acted as a supervisor in charge for the project. He supervised all aspects of the project and especially its planning, process, management, and general state. He also acted as a sort of mediator between the client and the team. There was no official technical supervisor for project, but since the client had implemented the features earlier and was familiar with the code the team could ask him for instructions.

5 Process and schedule

This chapter explains the process and schedule that were used in the project Muksis. It also describes how much time the project members spent on working every week.

5.1 Process model

The project was carried out using an agile process model that used multiple sequential two-week iterations. The beginning phase was called the 0-iteration. The topics of the project, workspace and personnel were introduced during it. The 0-iteration acted as a start-up and orientation step with no software implementation. After this, the iterations were performed in two-week cycles.

In each iteration the product was built further from the previous iteration. This ensured that the project was progressing steadily, and the participants got to inspect the results as early as possible. Short iteration intervals also helped the team to react faster to changes in project flow.

5.2 Schedule

Because of the Agile process model, a different timetable was planned for every iteration. At first only an overview for each iteration was written in the start-up phase. The final schedule for the next iteration was planned at the end of each two-week cycle and presented during the meeting.

5.2.1 Iteration schedules

Table 5.1 shows the main events of each iteration. The project encompassed seven two-week iterations, which made 14 weeks in total. There were no major changes in the iteration list comparing it to the one in the project plan until iteration 5, when the project had to add one extra iteration to be able to acceptably complete the project. The reason for this was an error with the DVB subtitles that was found relatively late in project. The bug was especially hard to trace since it seemed to appear almost randomly. The bug affected DVD subtitles so that there was an annoying delay in some

files. Richard solved the bug just before Christmas with a hint from the client. Also some of the iteration interval dates had to be changed, but every iteration completed in approximately two weeks.

5.2.2 Iteration steps

All tasks were decided iteration-by-iteration. In the meeting at the end of the iteration the project team reviewed the priorities of the tasks and requirements with the customer and decided which tasks the team would do next. After the meeting, in the internal meeting of project team, the project members were assigned new tasks and the time usage estimated. The whiteboard was used for recording progress during iteration. Previous schedules were used to more precisely evaluate this. If it seemed like some tasks used more time than planned in more than one previous schedules then the planned working time was increased. The same way the planned working time was shortened if it seemed like some tasks used less time than planned. Most approximations for the implemention times were tweaked the same way. The software production style varied a bit by the member of project and by the feature being implemented. Tommi seemed to use the typical typical design-implement-test sequence. Teemu and Richard commented that their style was more like a random code hacking and testing. The complexity of the MPlayer's code was clearly one reason for experimenting to be able to see what the code actually does.

5.3 Weekly working hours

The whole team's working hours were divided as the Figure 5.1 shows.

Soon after the subject of the project was introduced, the project team began to research the material available. The members could easily spend time on project since they didn't have many other on-going courses. The team progressed fast and the software seemed to work fine. This might be one reason why the during the middle of the project the working hours are dropping repeatedly. However when the testing phase began it revealed all the bugs that the team had missed. This again increased the time spent on the project. The reason why there are so few hours in week 48 is that most of the members attended an intensive course that was lectured in that week. Maybe if the bug testing would have started earlier we could have ended the project with more working hours and the graph would be smoother. Since the

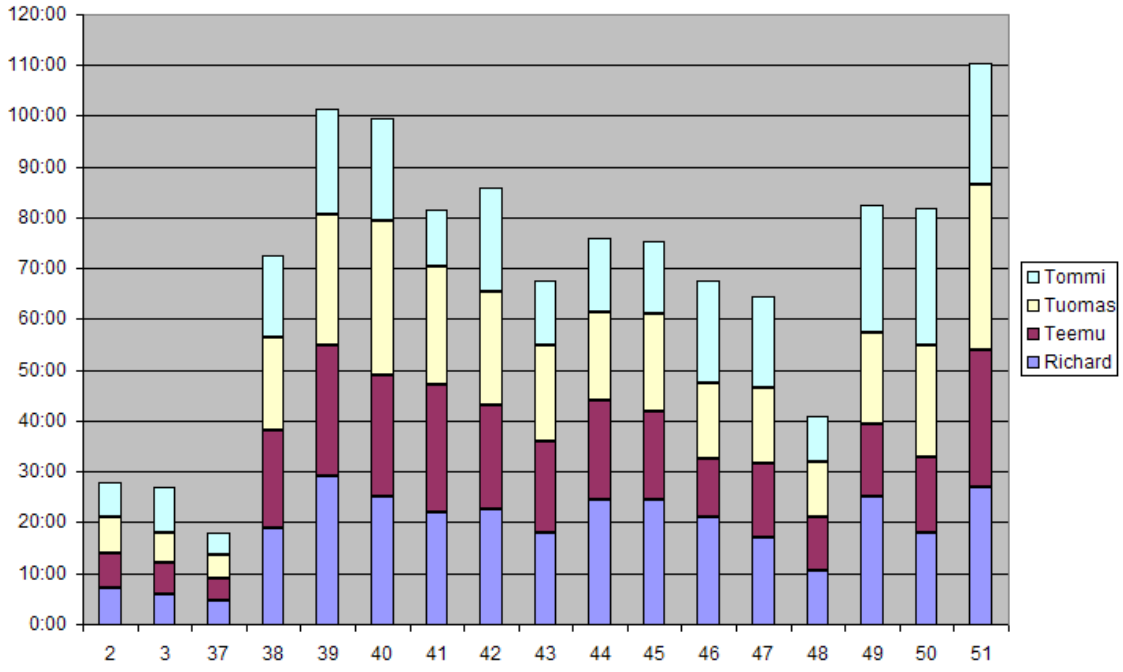

Figure 5.1: The project team's weekly working hours.

project started during the fall 2008 the weeks 2 and 3 in this graph present the two last working weeks in 2009 after the Christmas break.

5.3.1 Richard Domander

Richard's working hours were divided as Figure 5.2 shows.

Richard's weekly hours rose fast and he spent lots of time working in the project room researching the code and MPEG technology. His working hours continued steadily until the middle end of the project when he had to study for a few exams. At the end of the project his working time jumped back to near 30 hours per week since he had to write the application report and fix bugs. Since the project started during the fall 2008 the weeks 2 and 3 in this graph present the two last working weeks in 2009 after the Christmas break.

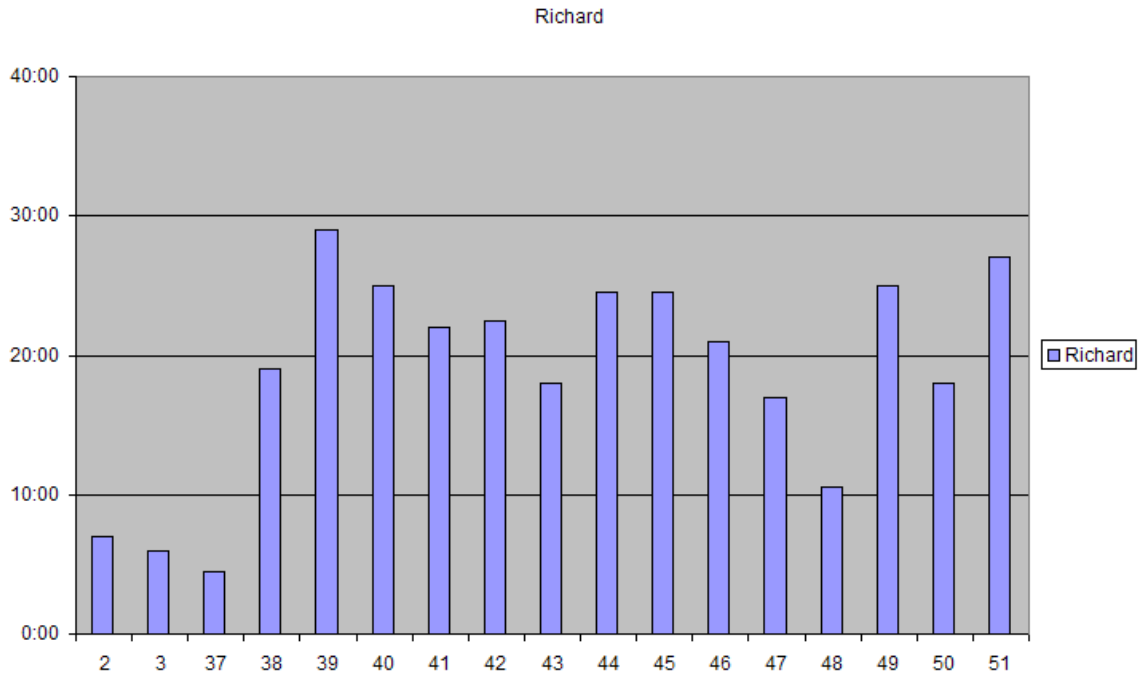

Figure 5.2: Richard's weekly working hours.

5.3.2 Teemu Nisu

Teemu's working hours were divided as Figure 5.3 shows.

Teemu's working hours were also rising fast at the beginning. He spent much time on researching and testing the code. He also wrote the testing plan and report during the second half of the project. Since the project started during the fall 2008 the weeks 2 and 3 in this graph present the two last working weeks in 2009 after the Christmas break.

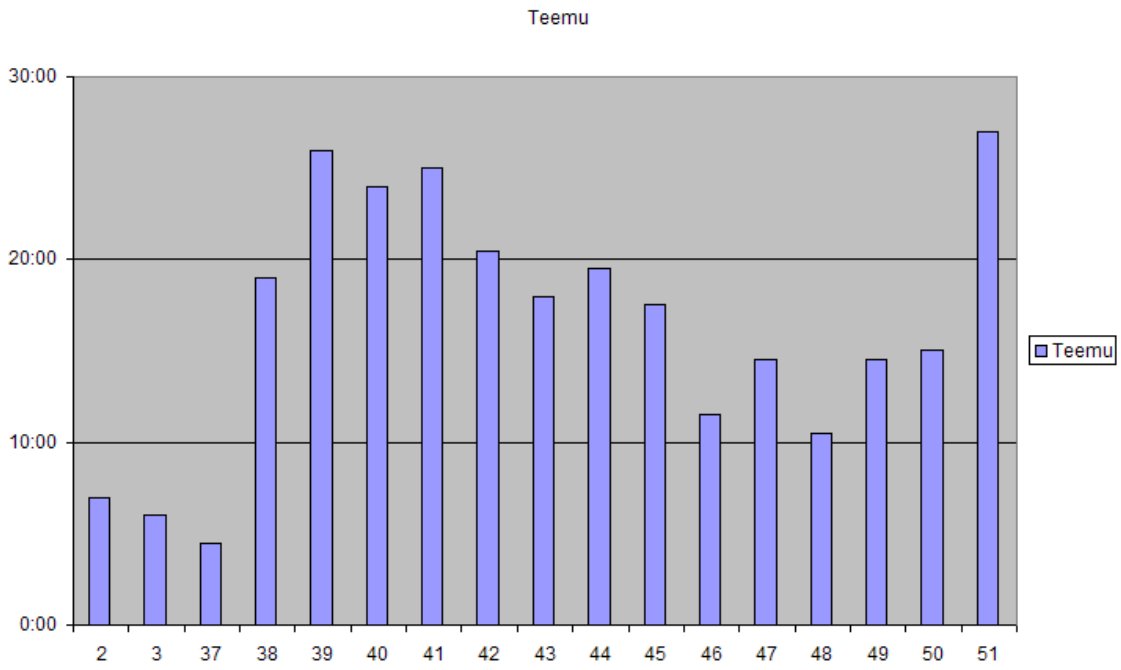

Figure 5.3: Teemu's weekly working hours.

5.3.3 Tommi Teistelä

Tommi's working hours were divided as Figure 5.4 shows.

Tommi's hours were relatively steady at the beginning of the project. There were a few spikes due to his working habits, however. At the end of the project he spent much more time on the project to even out his working hours with other members. Since the project started during the fall 2008 the weeks 2 and 3 in this graph present the two last working weeks in 2009 after the Christmas break.

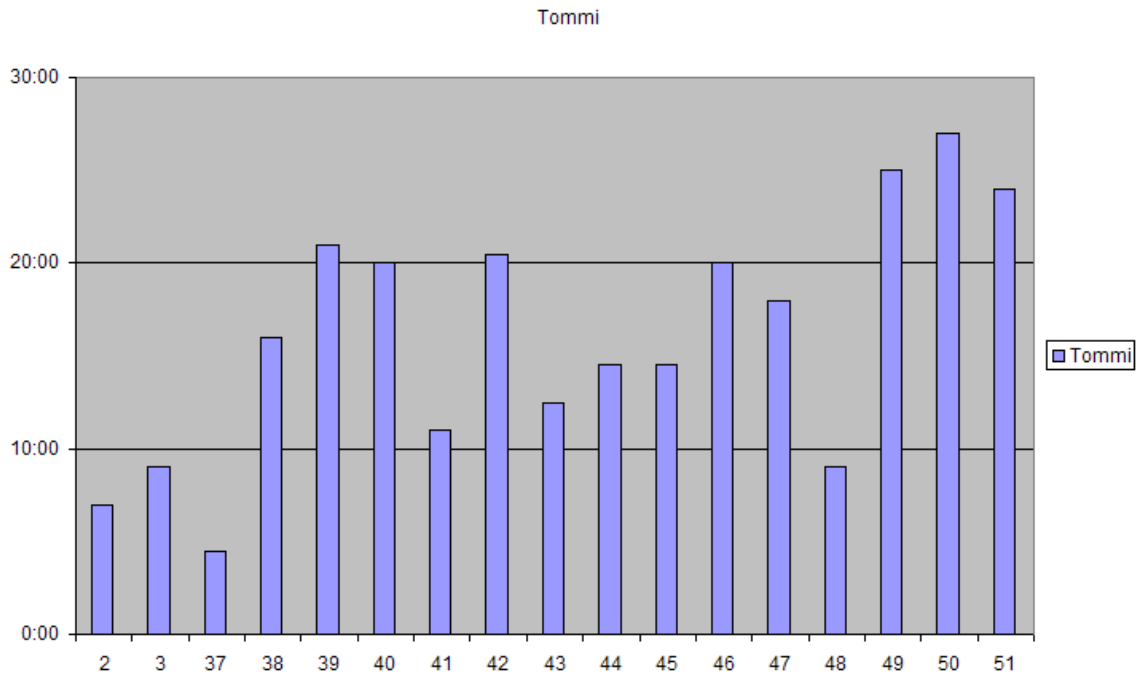

Figure 5.4: Tommi's weekly working hours.

5.3.4 Tuomas Mäenpää

Tuomas's working hours were divided as Figure 5.5 shows.

The weekly hours of Tuomas climbed fast but after that there was a steady downward trend until the middle of the project. Since the features that were going to be implemented were assigned to other members, he didn't have to work on coding but helped with testing the product. Then at the end of the project the project report required more work. Since the project started during the fall 2008 the weeks 2 and 3 in this graph present the two last working weeks in 2009 after the Christmas break.

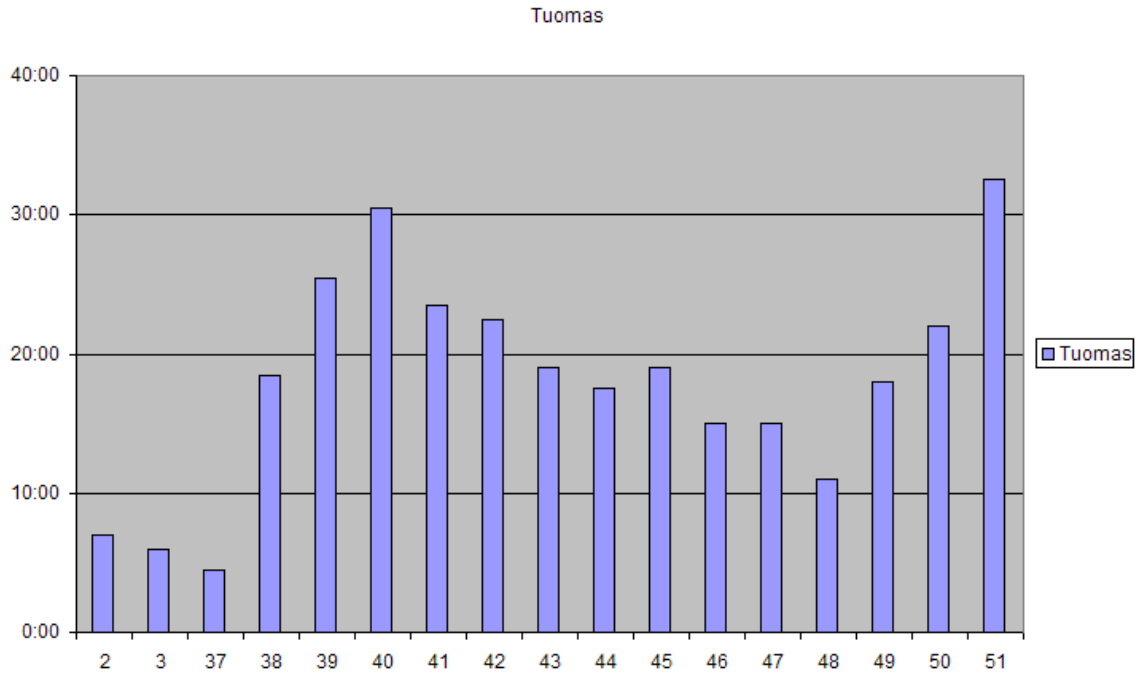

Figure 5.5: Tuomas's weekly working hours.

Table 5.1: Contents of the iterations.

Iteration	Duration	Interval	Most important tasks
0	2 weeks	17.9.2008 - 1.10.2008	Introduction to the project subject, research of specs and code and beginning to write the project plan.
1	2 weeks	2.10.2008 - 17.10.2008	Black frame detection's implementation, completing project plan, research of seek function and dividing it's implementation to tasks.
2	2 weeks	18.10.2008 - 29.10.2008	Seek function's implementation and starting to implement DVB subtitles, writing application plan. DVB subtitle implementation.
3	2 weeks	30.10.2008 - 12.11.2008	Testing the implemented features and writing test plan & report and starting to write the application report.
4	2 weeks	13.11.2008 - 1.12.2008	Completing the application report, testing the features in other environments and versions of MPlayer, project report's main chapters
5	2 weeks	2.12.2008 - 17.12.2008	Fixing severe bugs, making corrections to application plan and writing the rest of the project report.
6	2 weeks	18.12.2008 - 10.12.2008 and 7.1.2009 - 16.1.2009	Making final corrections to the project report, fixing all what is possible to fix, preparing the final patch and ending the project.

6 Tasks, workload and division of labour

At first the project was divided into different phases or steps. These phases were: peripheral course, project management, meetings, research, implementation and testing. To ease management, these phases were divided further into different responsibilities. These responsibilities were taken into account in every iteration when the tasks were assigned. The responsibilities also affected straight to the total working time of a member. Some of the features were more difficult to implement and had more bugs than others so they also needed more time.

6.1 Division of labour by phases

The overall division of labour is presented in the Figure 6.1.

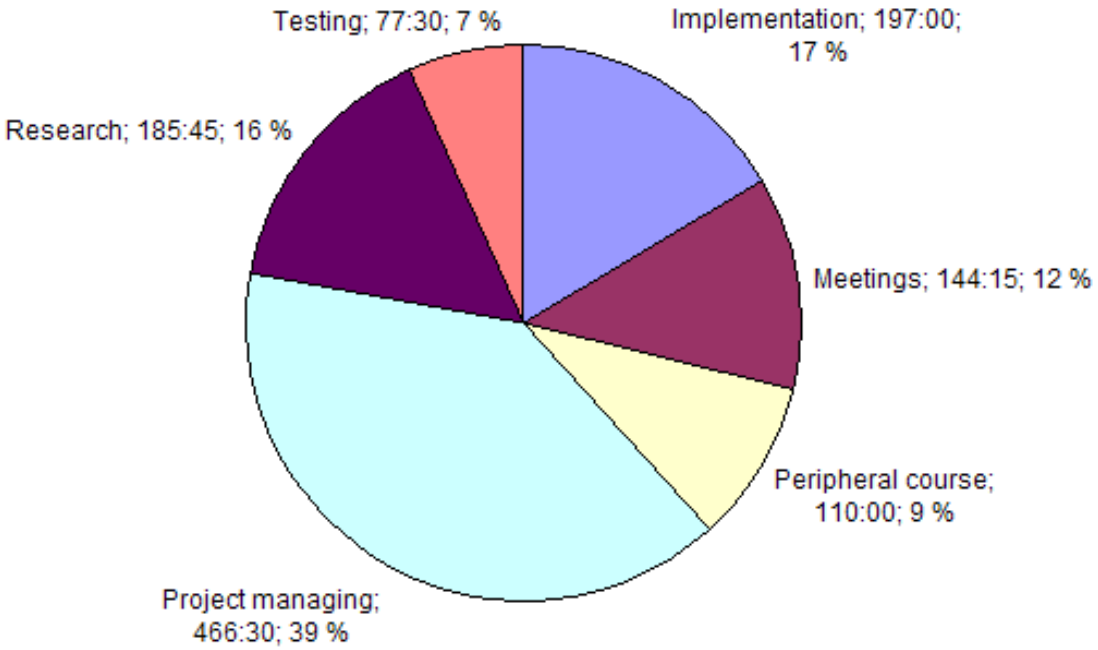

Figure 6.1: The division of labour in project by phases.

As Figure 6.1 shows, most of the time in project was spent on project managing and research. However, all the documents that were written were logged as project management and this added many hours to the phase. Researching the MPlayer code

was essential for understanding how to implement new features for the software. Studying the technologies involved in presenting MPEG video also needed lots of time. Agile software development also added some time to project management, since every iteration is planned separately during the project.

6.1.1 Richard Domander

Richard's division of labour is presented in the Figure 6.2.

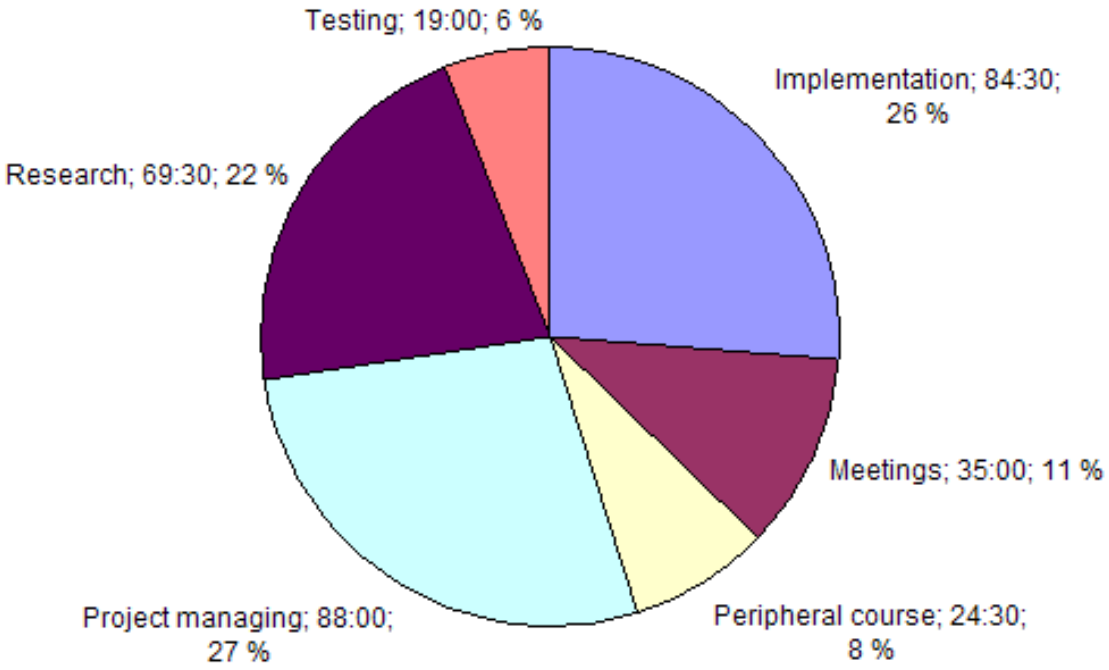

Figure 6.2: The time used for phases by Richard Domander

Richard used time quite evenly between different phases. At first he did research and implementation. He was also responsible for writing the application plan and report, which took quite a bit of time near the end of the project. Richard worked on the code for the DVB subtitles and black frame detection. Since code for DVB subtitles had most difficult bugs and Richard was responsible for fixing them, his working hours dilated towards the end of the project.

6.1.2 Teemu Nisu

Teemu's division of labour is presented in the Figure 6.3.

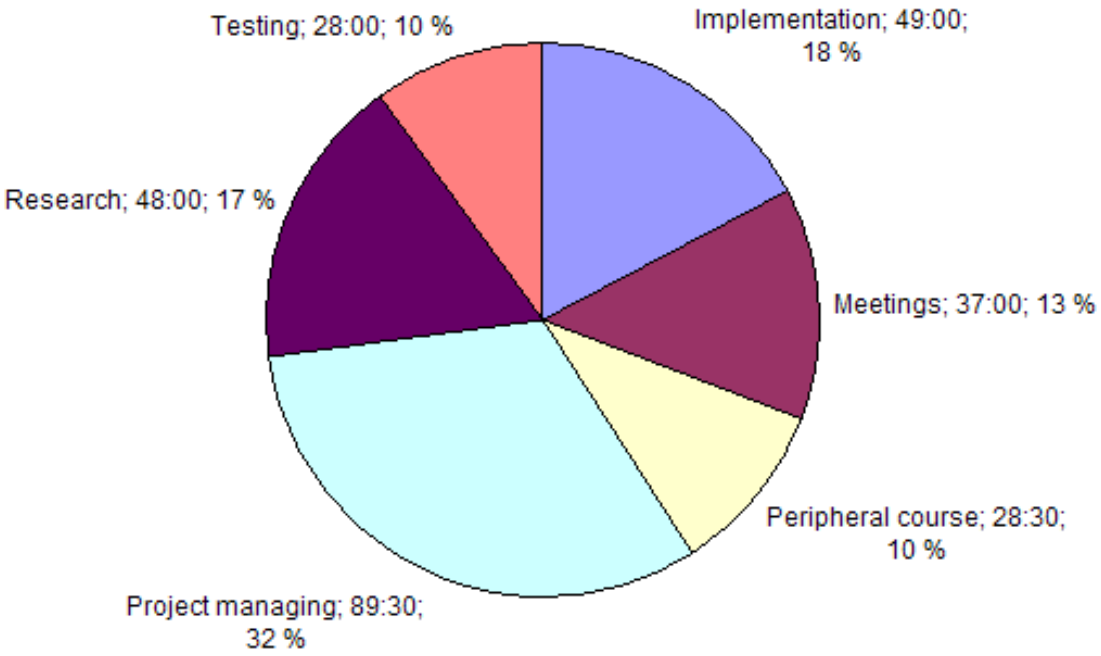

Figure 6.3: The time used for phases by Teemu Nisu.

Teemu wrote the testing plan & report and the *Risks* chapter for the project plan. This added time to his project managing phase. He also researched the code at the beginning of the project and did most of the testing.

6.1.3 Tommi Teistelä

Tommi's division of labour is presented in the Figure 6.4.

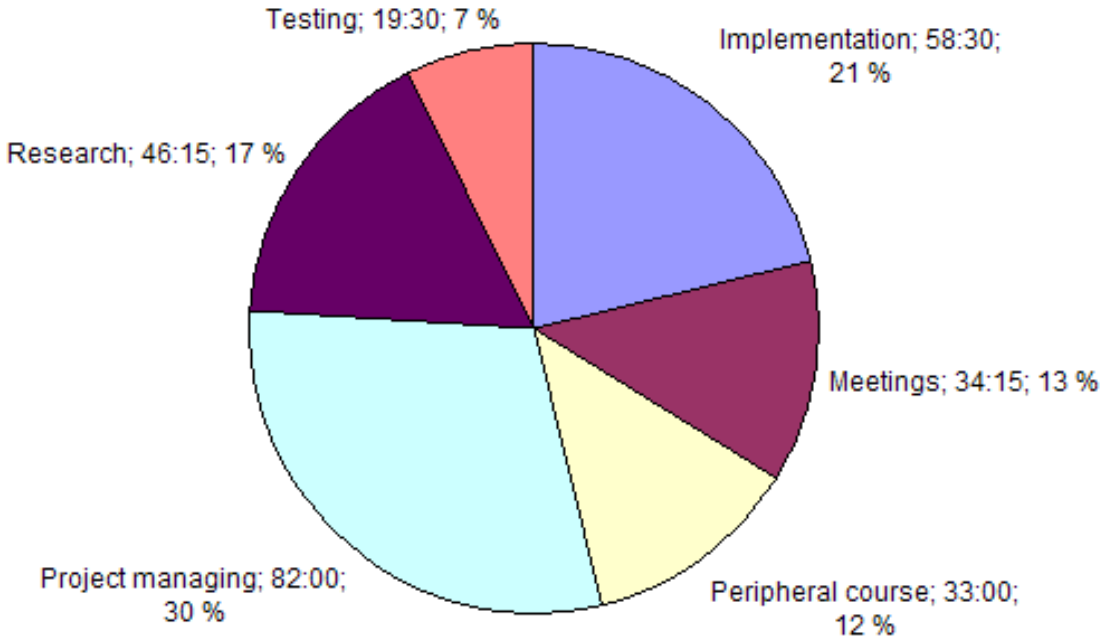

Figure 6.4: The time used for phases by Tommi Teistelä.

Tommi implemented the new seek function for MPlayer and later the audio filter. He also wrote chapters for the application report and for the project report, and was responsible for the team's presentations. He also researched the black frame detection code at the start of the project.

6.1.4 Tuomas Mäenpää

Tuomas's division of labour is presented in the Figure 6.5.

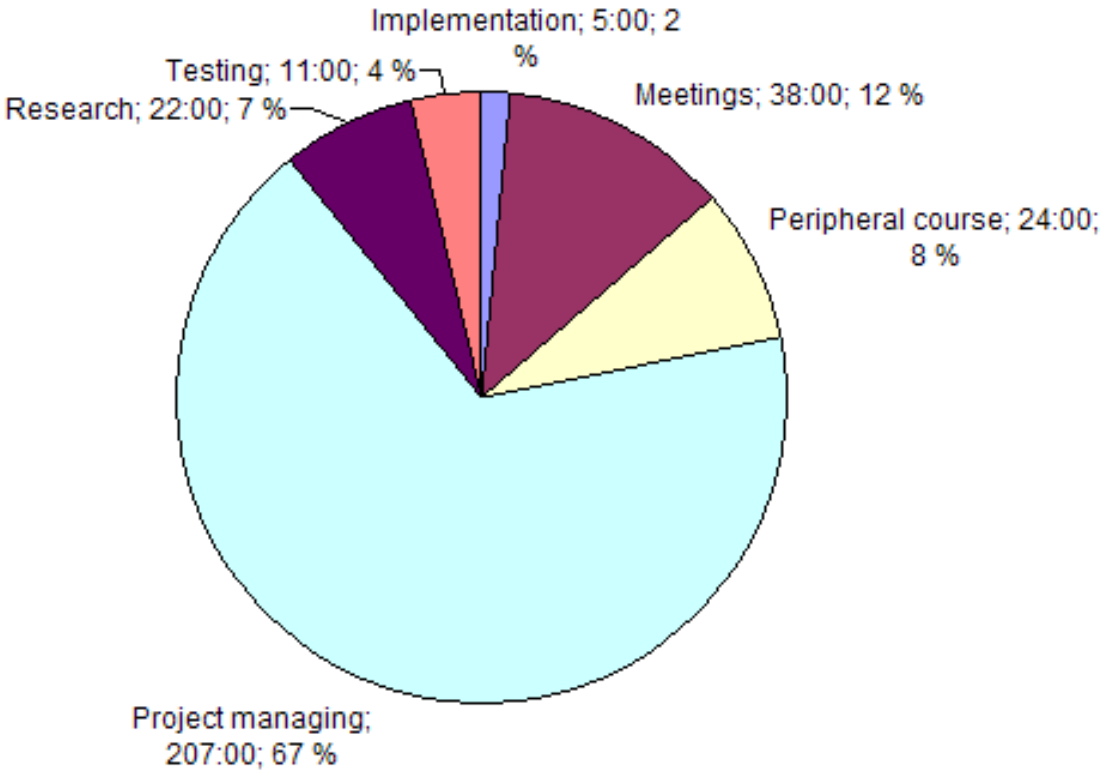

Figure 6.5: The time used for phases by Tuomas Mäenpää.

As the project manager, Tuomas mostly focused on tasks that involved project management. During iterations 0, 1 and 2 he wrote the project plan and researched the specs and code. At the end of the project he wrote the project report. Because there were so few features to be implemented he basically did no real coding at all.

7 Management methods

This chapter describes management methods of project Muksis.

7.1 Meetings

An official meeting was held between every iteration with all the stakeholders of project Muksis. Before the meeting the project team planned and prepared the material, which included a proposal on the next iteration's requirements and explained how the working hours were distributed in the previous iteration. In the beginning of a meeting the previous iteration was reviewed, and the tasks for the next iteration were decided. The chairman of the meeting brought and set up the needed equipment to the meeting room, and opened the session. The secretary prepared the agenda of the meeting, and took notes during the session. After the meeting secretary also wrote a report called minutes, which reviewed the main items discussed and any decisions that were made. All material was to be delivered two work days before the meeting, the agenda one day before and the minutes a maximum of three days after the meeting. The meetings were found really useful since the client had so much knowledge of the MPlayer and the team could easily ask questions about the code.

7.2 Communication

Communication was maintained with the e-mail list `muksis@korppi.jyu.fi`. In general it was only used to inform about the agenda and the minutes of the meetings and to send the hyper links to the material for all participants. Unofficial meetings were held at least once per week if there were no official meetings. These meetings were also found useful and more relaxed than the official meetings. The team could report the status and to plan what to do next without thinking too much about the agenda.

At the beginning of the project the supervisors had also delivered the contact information to every participant. Tuomas found phone numbers very useful since sometimes he had to get response fast for some issues. Most of the team's internal communication was maintained with informal meetings at the team's office. An

e-mail list `muksismafia.group@korppi.jyu.fi` was also established and used for internal communication between the project's developers. Overall it was not used as much as the list `muksis@korppi.jyu.fi` but it was useful for some small internal messages and we could use Finnish.

The project manager had the responsibility to act as a mediator between the team and the rest of the participants and to maintain communications within the team. Most of the communication outside the meetings with the client was handled with direct e-mails and with the project muksis's e-mail list. The project members could have used more direct live communication between with the client. This probably would have helped the team to solve some of the bugs faster.

7.3 Task management

After meetings the team planned the next iteration together by dividing tasks for each member using the project room's whiteboard. These tasks were usually decided in the meetings, but details were often worked out on the whiteboard. The description of the task and the estimated time it would take, along with the team members who would participate in it were each written on their own columns on the whiteboard. During the iteration the team attempted to estimate each task's progress with percentages written to the whiteboard. This could have be done more accurately. If the team would have updated the whiteboard more often then perhaps the total working hours between team members would differ less.

7.4 Time logging

The time usage of each iteration was logged into an Excel document. At the beginning the document was edited on a Linux machine, but the macros of graphs didn't work properly. The team decided that the project manager would log all working hours with the windows machine he used in the team's room and the other members would send their working hours to him. The working hours of each team member were written to the document's tables, grouping them by tasks. The time used in the peripheral course was also logged using the same Excel table. In the beginning of the iteration, the project team evaluated the estimated time needed to accomplish each task. This required the team to have internal meetings weekly to divide tasks

and to estimate the time requirements for them. In the end of the iteration the time log was used to analyze how well the plan held during the iteration.

7.5 Responsibilities

To make the project work smoother, all work was divided into sets of responsibilities. Each team member was assigned a primary responsibility. The peripheral course's lectures, group works and presentations were handled as a separate responsibility belonging to all team members. Each person was in charge of monitoring the progression of his responsibility but didn't necessarily handle it all by himself. The division of responsibilities worked quite well, but it wasn't followed strictly. If it looked like that some important task was going to take more time from the member, then someone else was usually arranged for the job. However there was no need for any major changes in responsibilities.

7.6 Documentation

The project's documents were processed with the LaTeX typesetting software using the department's document templates and named using the format `document_name-version.pdf`. All documents were published in PDF format and stored in the public hard drive mount created on the network of the University of Jyväskylä, and are available for all. At the end of the project these files will be burned on a CD-R disc and placed into the project folder. In addition, meeting agendas and minutes are were handled with the project management application Trac, where they were easily editable and accessible to all participants. All complete plans and reports were signed by the customer, the project manager and the supervisor.

7.7 Version management

The team used Subversion for version management. Every official document which had a version number was named using the format `document_name-a.b`, where a is the major number of version with only values of 0 or 1. b is the minor number of version and is an integer starting from 1. For

example after version number 0.9 comes 0.10 not 1.0. The number 1.0 indicates that the document is complete. At the beginning the SVN was not used and the members experimented with the source code placed on their computers. Soon after the lectures of version management all members agreed to begin using SVN and all the code was placed there. All Excel documents and TeX files were also placed to the SVN. The SVN clearly made the implementation smoother and after Teemu noticed the feature to create patches straight from the SVN it made our work even faster.

8 Risks

Unexpected events during the process could have prevented it from succeeding. This chapter goes through the risks that were estimated in the project plan and analyzes if they came true and how they affected the project. The risks are defined in Table 8.1. The probabilities and effects of each risk are evaluated on a scale of low-intermediate-high.

Table 8.1: Risks of the project.

Risk	**Probability**	**Estimated effect**	**Realized effect**	**Came true**
Client's availability	Low	High	-	No
Supervision	Intermediate	Intermediate	-	No
Inability to participate	Low	High	Low	Yes
Human relationships	Low	High	-	No
Inexperience	High	Intermediate	Intermediate	Yes
Programming skills	Low	High	-	No
Scheduling	Intermediate	Intermediate	-	No
Communication	Intermediate	High	Low	Yes
Language	Intermediate	Low	-	No
MPlayer development	Low	Intermediate	-	No
Working environment	High	Intermediate	Low	Yes
Complexity	High	Intermediate	High	Yes

8.1 Client's availability

The client was available the whole time during the project. The team communicated with him by e-mail and the state of the project was discussed in every meeting.

8.2 Supervision

The supervisor provided useful information about managing the project and was available all the time. In the end the project didn't get a technical supervisor, but since the client had much insight about the subject there wasn't really any need for one.

8.3 Inability to participate

On some occasions the members weren't all able to participate as planned, but this had little or no impact to the project. Near the end of the project there was an intensive course about 3D game programming and the members attended it. Just before the intensive course some bigger bugs had emerged, but the course didn't seem to interfere with the project's progress. Since the bug was found just before the course we did not have much time to react. It just showed as an extra work in the next week.

8.4 Human relationships

The group members got along well and there weren't any big issues, although the team could have spent more time together in the project room.

8.5 Inexperience

The project would probably have gone smoother if the members had more experience. For the project manager planning accurate working hours felt a bit hard at first. Also writing the long documents in foreign language took longer than expected. Tuomas tried to react this by starting writing the project report sooner but it still took longer than he planned. Since the source code was messy and large, it was hard task for relatively inexperienced coder to know what to search and where. This was reacted by adding more hours to the schedule's plan. Especially with the DVB subtitle's delay bug it almost seemed like we might not repair it in time, but it got handled eventually.

8.6 Programming skills

The team members had at least basic knowledge of writing C. This seemed to be enough, since there weren't any real deadlocks during the iterations.

8.7 Scheduling

Some tasks like writing the plans, reports and repairing seek code and DVB subtitle's delay bug were a bit late, but the overall impression was that the schedules held quite well.

8.8 Communication

There were some small issues in communication, especially at the beginning of the course. The team could have worked more efficiently if they would have communicated more with the client. However the impact of these problems was minor. The team reacted to the lack of internal communication by creating the muksismafia mailing list, so we could communicate in our main language.

8.9 Language

Writing the documents and communicating in meetings in a foreign language was a challenge. Still there weren't any real cases where the language would have negatively affected the progress of the project.

8.10 MPlayer development

The members estimated that the MPlayer could evolve so much during the project that the implemented features wouldn't work in the newest version. At least before the December most of the code members had written worked by just merging it to the MPlayer's SVN.

8.11 Working environment

Some of the application tools installed on the computers didn't work as expected. This created some extra tasks trying to solve the problems and finding alternative ways to do the work. The team tried to compile the software with the Code::Blocks IDE, but they soon found out it is too difficult. After noticing this the IDE was only used as an text editor since it had a great set of tools such as highlighting for writing the code. The Linux machines used Fedora as their OS. For some reason the Gnome desktop did not work as it should have and stressed the team members.

8.12 Complexity

The MPlayer is complex and large software. To make things worse, the source code in general is quite messy. The complexity seemed sometimes overwhelming and the code needed lots of experimenting before the team members could understand it. The team reacted to the complexity by studying all the technical material the client delivered to them. Over the time when the MPlayer felt more familiar the implementation got also easier and project resources allowed to fix the bugs.

9 Personal experiences

9.1 Richard Domander

When we started the project I didn't doubt its chances of success as much as I did my own. I had little experience of larger projects before delving into the code of MPlayer. I knew a bit of C, but Linux environment was unfamiliar to me. Surprisingly, with hard effort I began to understand how MPlayer functions. After dozens of hours spent researching, I understood the source code enough to know which parts we needed to alter to implement the required features. I also knew how to improve the code in Weber's old patch.

New bugs (see application report) kept showing up especially with DVB subtitle support, which was my main responsibility. Even though it was frustrating at times, overall I was glad that I could concentrate on implementation. Solving a bug often seemed an insurmountable task at first, because I didn't have any idea what to do, but with time I managed to solve it. My coding could be described as "trial & error hacking". I just bang the keyboard, and change things guided by my magical techno intuition, without ever truly understanding what I am doing. Often I began to understand the solution to the problem only after I was finished. This kind of implementation is rather slow. I should begin to think more before acting, to plan my coding. I should also be less stubborn, and admit defeat more readily when it's clear my initial idea won't work. Note to future developers of MPlayer: you can't ever research the code enough, but some research is best left to the implementation phase when it's clearer what you need to change.

During the project I started to get annoyed by the amount of "meta work". I felt too much time was spent writing various documents and sitting in meetings in relation to time spent for actual coding. But I know that the main purpose of the software project is education, so it's understandable that some things are done just for the sake of doing, and thus learning them. At least my English got a lot more fluent because I had to write so much stuff in it. I learned something about process and project management, but I feel the process could have been even more "Agile", i.e. less management overhead – our project leader didn't get to code at all! But I know our project was a special case, which was difficult to fit into the existing process & project models.

9.2 Teemu Nisu

When I saw the subject of our project for the first time, I got both excited and worried. The subject was very interesting but on the other hand I had no idea about the implementation. I had some experience with C++, but something as huge as MPlayer coded with plain C was totally new to me. In the beginning I had to use a lot of time for researching MPlayer's source code to get some kind of grip of it. The old code provided by Matthieu helped a lot in finding the key parts to focus on. Most of the code also fit quite well into the newer versions of MPlayer giving us a good basis for development. After we got the features working to some extent, we were relieved and felt that we might even finish the project early. So we eased off a bit. Then the testing phase came in and we realized that the project was not so close to completion after all. Since the reporting phase was also closing in, we had quite a busy time before Christmas.

It became clear from the start that things like planning, documentation and having meetings would actually take more time than programming. It was partially caused by the nature of the project, since there was not so much actual programming required. Implementing the features was more about porting code and experimenting with things rather than writing things from scratch.

What I expected from this course was getting an overall impression of software projects in practice. So far it has met my expectations quite well. Despite the fact that the subject of the project was quite special, it has given me a lot of valuable experience that will surely be useful in the future. Especially things like presentations and meetings cannot be learned by reading books or listening to lectures.

9.3 Tommi Teistelä

When I found out that the original project planned for us had been pulled out at the last minute and we would be getting a new one as soon as it could be arranged, I was a bit worried we'd end up doing some rushed, poorly thought out project that got the "second place" for a very good reason. Now, I can certainly tell why the one we got had been waiting on the "second place" for a while, but I'm actually glad we got to work on this – I'd wanted to look at a free software project for some time, but hadn't yet found a good reason to make me pick one and focus on it over others. There are a lot of them that could use some improving...

I mainly worked on the MPEG TS seeking and audio filtering features, which were very different from each other. With the MPEG TS seeking I was tweaking the internals of a completely undocumented software component to do something it wasn't built to do, while the audio filtering was mostly separate from any existing code and easy to implement from scratch. The audio filter took me about two evenings, while the seeking feature kept me busy through several project iterations figuring out which "feature" of the code I hadn't touched I'd triggered this time, which variables actually held meaningful information at any given point, etc. I don't know if I've learned anything about the C language I didn't already know, but I definitely have gained a new appreciation for proper documentation and to the way access to structured data tends to work in more object-oriented languages.

Like Richard, I was a bit surprised by the overall amount of time we spent working on the project's management – meetings, writing reports and so on, but it was just another aspect of the course and one topic I certainly learned something new about. As much as I've usually disliked writing stuff over coding, my experiences with working on the MPlayer code led me to value the occasions when I just needed to write plain (English) text. As usual, I probably spent at least as much time worrying about the quality of my work as writing it, but I've come to accept that as part of my creative process.

9.4 Tuomas Mäenpää

As a whole the course felt interesting and challenging. I didn't have any real experience of leading a project before the course and I had some doubts about my skills for this. I knew at least the basics of the C language but I was kind of expecting to do a web based application before the subject was revealed. I have some real life experience of working in large scale software project so I had some insight what to expect. The Agile process model seemed quite interesting and I believe I have gained much valuable experience during the software project. I have always liked practical or down-to-earth style of working, so this course felt especially refreshing after all the theoretical courses.

In the beginning of the project there were clearly three key features that our team was going to implement. This situation let me assign these features to other members and concentrate on planning the project and writing the project plan. I probably should have tried to arrange even a bit of coding work for myself since later

the work had clearly been divided for the other members. After this it soon felt like there was no need to dive into the source code since the features progressed smoothly. I think this in a way or another made us shift our working pace a gear down. After we began testing we realized how much work there still was and because the end of the project approached fast we had to work faster again.

10 Bibliography

[1] MPlayer documentation; Appendix D. History, referenced 10.10.2008
http://www.mplayerhq.hu/DOCS/HTML/en/history.html

[2] University of Jyväskylä; KIEPO termipankki, referenced 29.9.2008
http://www.jyu.fi/hum/laitokset/solki/tutkimus/
projektit/kiepo/termipankki/opintopiste/
?searchterm=opintopiste