Final Report

Group Mavis

Helsinki 5.5.2006
Software Engineering Project
UNIVERSITY OF HELSINKI
Department of Computer Science

Course

581260 Software Engineering Project (6 cr)

Project Group

Kimmo Holm

Juho Julkunen

Rami Järvinen

Janne Laukkarinen

Joel Linden

Jan Wagner

Client

Fabio Donadini

Tomas Kohout

Project Masters

Juha Taina

Joni Salmi

Project Home Page

http://www.cs.helsinki.fi/group/mavis/

Table of Contents

1 Introduction

This document describes the work that went into the student project of bug-fixing Ikayaki, the user interface for the SQUID magnetometer and building a new analyzing software we named Mavis. The clients were Lauri Pesonen with his assistants Fabio Donadini and Tomas Kohout from the Department of Geophysics.

The Department of Geophysics uses a SQUID magnetometer for measuring the magnetism of rocks and meteorites. Ikayaki is a program that works as a user interface for controlling the SQUID. Our job was to fix some bugs in Ikayaki as well as build some new features into it. An additional task was to build a software for the analysis of measurement data measured by SQUID and Ikayaki. This analysis software is called Mavis, short for Magnetism Visualization.

The project took place from 16.01.2006 to 07.05.2006.

1.1 Organization

Name	Role	E-Mail
Kimmo Holm	Project leader (Ikayaki)	kimmo.holm@helsinki.fi
Juho Julkunen	Requirement specification, Project leader (Mavis)	juho.julkunen@helsinki.fi
Rami Järvinen	Testing	rami.jarvinen@cs.helsinki.fi
Janne Laukkarinen	Software architect	jlaukkar@cs.helsinki.fi
Joel Linden	Documentation	jlinden@cs.helsinki.fi
Jan Wagner	Coding	jwagner@cc.hut.fi
Tomas Kohout	Client	tomas.kohout@helsinki.fi
Fabio Donadini	Client	fabio.donadini@helsinki.fi
Juha Taina	Course Manager	taina@cs.helsinki.fi
Joni Salmi	Instructor	jsalmi@cs.helsinki.fi

1.2 Project

This section discusses the progress of the project from its beginning to the end.

1.2.1 Work Flow

In this project we used two different process models. Iteration process model was used in developing Ikayaki and waterfall model was used when we implemented Mavis. Here is how each of the stages in the process went.

1.2.1.1 Project Plan

Ikayaki - Kimmo

When I was assigned - Basically, since nobody really volunteered - the project leader job, I knew I would have major problems with project plan. I was right. I'm terrible at planning things in advance, being more of an extreme programming-kind of guy. So, I went through my options and decided to go with iterative model with bit of an extreme programming added for touch. I thought my plan and

especially timetable was great - until the dreaded feedback. Oh well. Regardless, after carefully considering the valid and valuable feedback, I believe I was able to come up with an adequate project plan. Or so I thought, anyway. In the end I don't think we entirely managed to stick to the plan at least for any given set of iteration but I do believe we kept to the spirit of the plan - which was iterative, extremist approach to programming. We planned, programmed and released. Far fewer iterations than I had hoped for but few nonetheless.

Mavis - Juho

I took over as project leader for development Mavis, since Kimmo had too many demands for his time. For Mavis we didn't have time for iterative model, but rather followed waterfall model, with distinct phases for requirement gathering and analysis, system design, and implementation. Time frame was extremely short, however, and we rushed through analysis and design to get to the implementation phase. Little more time spent on the drawing board would have been preferable, but time was short. As it was, we managed to implement the absolutely essential requirements to make the software usable, but little else. Customer was happy, though anyone doing follow up development might not.

1.2.1.2 Definition

For Ikayaki requirements definition was done concurrently with implementation, with individual features assigned as their requirements were sufficiently established. Some of the requirements were bug fixes, some new features. Since we working on an existing software, for many of the requests requirements were already fairly well established.

Requirements definition was basically done in two phases. First phase was scoping out the requirements client had for the software and having them prioritized. Second phase was requesting more detail from the client as needed. In few cases this needed a few iterations. Once the team considered there was sufficient information to implement a requirement, it was assigned to a member of the team.

With Mavis there was no ready-made blueprint, and the first phase was establishing a broad picture of what the analysis software was for and what was expected of it. Based on this knowledge it was possible to create an initial list of requirements and a GUI mock-up, which was demonstrated to the client. Once client approved the GUI design and feature list, requests were prioritized.

Some of the requirements involved calculations which required detailed instructions from the client. Since time was short, some of these details were ironed out while the implementation was already under way. Additional delay was introduced when the client with most knowledge of the analysis software left country for two weeks. Still, in the end, the priority one features selected for inclusion were all defined sufficiently to merit inclusion.

1.2.1.3 Design

The architecture design of Mavis was primarily done by Janne. Detailed design was done by whoever was responsible for the particular class or component. GUI design was done by Juho and the UI Designer in IntelliJ IDEA was used for generating the code.

The design document was done quickly so that coding could begin as soon as possible, since we didn't have much time when we finally started the production of Mavis. Many things in the design document were left open, and team members had to come up with their own design decisions. Consequently, the design document and the UML class diagram in the wiki were not kept up-to-date most of the time.

1.2.1.4 Production

Coding of the bug fixes for Ikayaki started as soon as we had received the requirements. First changes were small and mainly cosmetic. As soon as the team had become more familiar with the pre-existing code, we began making bigger changes to the code. In the end, our team had learned a lot of the inner workings of Ikayaki, but some parts of the code may still remain cryptic to at least someone in the team.

In Mavis, coding began as soon as a basic architecture and classes had been designed. A major part of new code was coded during IRC sessions where all or most team members were present. Some changes to the design were made at the same time. It was really rewarding to see our little program "come to life" during these sessions.

1.2.1.5 Testing

Testing with full code coverage was planned at first, but in the end we didn't have time to accomplish it. JUnit didn't suite in our needs (see debriefing section). Only new features and changes to old code were tested by responsible programmer.

The SquidEmulator was used in testing Ikayaki's serial port traffic and degausser related bug fixes. Most of other testing was performed through application's GUI. Testing of Mavis also was done mostly by doing GUI use case tests. Reasons for that are mainly the same as with Ikayaki.

During Ikayaki's development process several test versions were given to client for testing in action. Client gave us valuable feedback of bugs and other lacking things.

1.2.1.6 Installing

Fresh development versions of both Mavis and Ikayaki were regularly installed on the customer's machine during the project. Installing was largely handled by Juho and the customer. Installer exe's adapted from the earlier project's Nullsoft Installer (NSIS) and JSmooth project files were assembled by Jan.

In retrospect, the configurability of NSIS was severely lacking. For this reason, possible future projects derived from Mavis or Ikayaki should consider using the free InnoSetup instead.

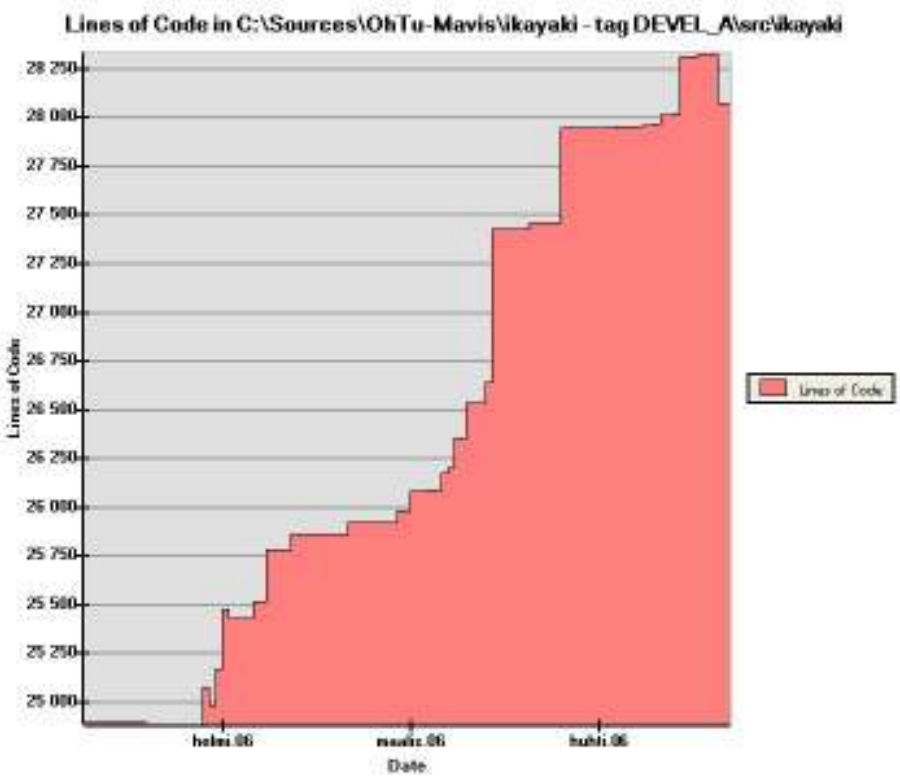
Final release versions were installed by the customers themselves.

The final binary installers for Ikayaki 1.1 and Mavis 1.0 will be publicly available to 3rd parties, too, in addition to the GNU GPL source code. More detailed setup instructions are provided as separate .txt files together with the installers. Installing Mavis is straightforward, as is installing Ikayaki 1.1 on a new machine, however an upgrade from Ikayaki 1.0(1) to 1.1 requires some care. This is documented in the setup instructions.

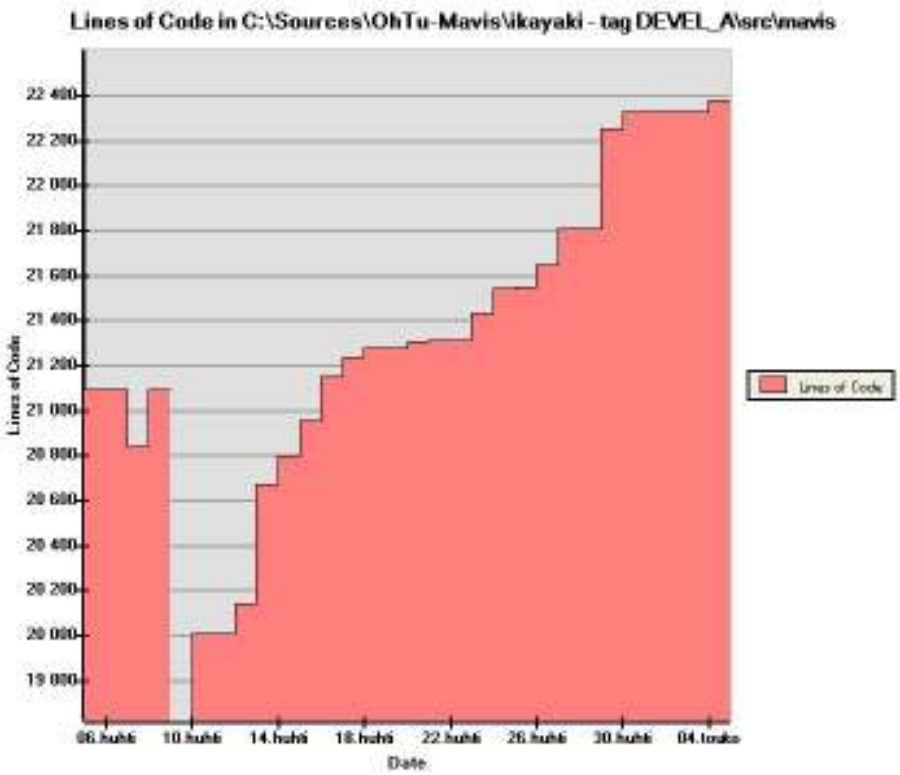
1.2.2 Amount of Work

The estimate for the size of the projects was 2000 to 3000 new lines of code for both the Ikayaki bug fixes and the analyzing software. With the estimate of 1000 to 1500 lines of code for the bug fixes, the estimate for the analyzing software Mavis was 1000 to 1500 lines of code. It turned out that we underestimated the group productivity.

According to CVS statistics generated with a trial version of CodeHistorian, Ikayaki had around 25000 lines of code when the team started the process. At the end of the process Ikayaki had 28300 lines of code. Therefore the amount of new code was approximately 3300 lines.



The final figure for Mavis was 22400 lines of code. Most of the code was either taken straight from Ikayaki or heavily modified for a new use. The amount of new code is around 3600 lines.



These statistics can also be found in http://www.cs.helsinki.fi/group/mavis/plots/

1.2.3 Features Unaccounted For

1.2.3.1 Ikayaki

Feature: Config information and history data recorded in separate files
Reason: The rationale was such that there were problems when the config files were copied to another system, where the directory and file structure were different. Since this wasn't common the feature wasn't considered a high priority.

1.2.3.2 Mavis

Feature: Automatic calculation of magnetic components
Reason: When we started working on Mavis, we realized we had to cut some features in order to deliver a working program. The client provided priorities for all the features and this was one of those with a lower priority. Since we didn't have the available time, we had to cut this feature along with some others.

Feature: Exporting to PDF
Reason: The feature was considered a low priority since it is possible to create PDF documents by using the printing feature and some available software to print to a PDF document. Exporting directly from the program to a PDF file would have required too much time and work.

Feature: Difference vector stereo plot
Reason: The client considered this feature to be a low priority, so it wasn't worked on. Probably wouldn't require much work unless it had to look radically different from the current stereoplot.

Feature: Calculating End Vectors
Reason: A low priority requirement which nobody worked on.

1.3 Debriefing

1.3.1 Overview

At the beginning of the project, it was a bit unclear what our role would be in the development of these programs. Many of the bug-fixes and new features looked easy and quick to code, although the preexisting code looked intimidating and most team members were unfamiliar with the theory behind the complex calculations. The planned analyzing software seemed to be the biggest work.

As the team was fixing bugs, receiving new requirements from the clients and then fulfilling those requirements, it started to look like the project was a piece of cake. Especially when the team foolishly thought they wouldn't have to build a new analysis software as long as the old software previously used to analyze measurements worked on a modern platform. When it was decided that the easy solution wasn't enough, the frantic design and implementation of the Mavis analysis software began.

In the end the team managed to build a working program that fulfilled all the higher priority requirements the clients had come up with. The client was happy with the software and the team feels proud of its accomplishments. We all learned a lot about the field of geophysics, working with preexisting code and the features of Java as well as numerous tools.

1.3.2 Project

What the project team thinks about the following:

- Meetings:
  Regular meetings were very important for keeping track of the progress of the project.
- E-Mail:
  Good for notifying everyone of important stuff, since not everyone was using IRC all the time. Also an important communication channel between the team and the clients as well as the team instructor.
- IRC:
  Not that useful in the beginning, when all team members didn't use it. Later when most team members started using IRC daily it became critical, especially during the development of Mavis. Big thanks to the team leader of the previous Ikayaki project, who was always hanging around on the IRC channel and was a great help in understanding the inner workings of the source code and everything.
- Wiki:
  A good way to keep documents and instructions and stuff up-to-date. Easy to learn and to use.
- Forum:
  We didn't use it as much as we thought in the beginning. Team members were probably more familiar with IRC and E-Mail.
- Java and Swing:
  Powerful and theoretically quite easy to program, except that often deep digs through the java api documentation are required. At project start, not everyone was up-to-date with the current java 2 api and java programming trends. Nevertheless this did not have any detrimental impact on product quality nor did it delay the project schedule.
- CVS:
  It was working quite fine after everyone got it working. The main problem was that the university servers did at times add severe lag to CVS. Using a dedicated CVS server would have helped. Exporting a clean summary (changelog, statistics) from the CVS updates proved to be tricky.
- IntelliJ IDEA:
  A pretty good Java IDE. Especially the UI Designer is neat. A bit slow though, takes ages to load, compile and use CVS, annoyingly freezes during GUI compile, and has a lot of features we did not really use.
- JUnit:
  It might have its specific uses, and it comes quite nicely pre-integrated into IDEA. However it was next to useless for testing the internal class state (state machine, if you will...) and private class methods, as well as any public methods that use inputs more complex than the basic data types, and useless for testing classes that interact with hardware e.g. serial ports. Some of the JUnit testing tasks were actually easier to perform manually through the application's GUI.
- Nullsoft Installer NSIS:
  While WinAmp is a good Nullsoft product, NSIS is definitely not. At a late stage it became apparent that the NSIS setup file include/exclude logic is fundamentally flawed. Conclusion: could be troublesome to use for future installers, avoid at all cost. Use InnoSetup instead.

1.3.3 People

1.3.3.1 Clients

Clients were very active during the entire project, they always found time to answer questions about requirements, and very conscientiously tested out intermediate versions of the Ikayaki software as they were released.

Some of the clients were perhaps even a little too involved. Possibly due to previous experience of programmers clients sometimes discussed requirements on too concrete and detailed levels, when a more general goal was wanted, sometimes even offering suggestions on the coding.

1.3.3.2 Instructor

Special thanks to our instructor, who occasionally stepped in and assumed the role of the project leader.

1.3.3.3 Team members' evaluations of themselves

Kimmo Holm

Having two jobs alongside with the project really cut into the time I was able to allocate for this project. After transferring project leader-responsibilities to Juho I was able to concentrate on programming only and time management became slightly bit easier. In the end, I believe I was able to contribute in a meaningful way to the project.

Juho Julkunen

I started the project as the requirements engineer, and quickly realized getting meaningful requirements from a client and communicating them to the team is a full time job. I took me a while to get the hang of it, but I feel that overall I did a good job. Since customers felt they got the program they wanted, I trust I was successful enough in the role.

For the latter part of the project I also took over as the project leader. That was a lot of pressure, and I'm not too confident that I did a very good job of it. Since the project was finished in time and only slightly over budget, and the customer ended up with software they were happy with, I suppose I wasn't a total failure.

Rami Järvinen

Somewhat lazy start, but got more involved in things later on. Anyhow I did many hours of work and produced good stuff. I'm quite content with my efforts.

Janne Laukkarinen

I got a bit lazy somewhere near the end and some things I was responsible for could have been done a bit earlier. Still I did a lot of work and I'm satisfied with my part of the project.

Joel Linden

I had only one full-time job (unlike Kimmo and Jan) alongside with this project but it was enough to kill some of my enthusiasm at least during the last month or so. Yet still I'm quite happy for how the project went and I feel that my contribution had some impact on the project results.

Jan Wagner

Similar as for Kimmo, two jobs cut the time available to donate to the project. Despite sleep deprivation ;-) I believe I managed to contribute a bit too on e.g. IRC, and to stuff in quite a lot of

productive work plus a few late night on-site coding sessions into those 140++ total hours. Which probably is because, unlike other unfortunate members of our team, I wasn't "harassed" with writing much any documentation, thank god for that ;-) Slightly lazy at the start, more committed later on, and for some odd reason most productive/fast during simultaneous #Mavis IRC sessions. Overall, I'm quite satisfied.