Chapter 1

Pre-project report (without appendixes)

1.1 Goals and constraints

Background

Skolelinux is an organization aimed at creating a cheap and simple Linux distribution for primary and secondary schools. It is used in more than a hundred schools already and the goal is to cover as many schools as possible.

To reach this goal they need software to cover the different needs the students have in their education. One such program, a teacher at a secondary school in Norway has asked for, is an application for creating stop-motion animations.

Effect goal

The effect goal is to offer an easy way for students at Skolelinux schools to produce stop-motion animations.

Result goal

A fully functional program students at primary and secondary schools can use to create stop-motion animations. The software should be simple to use and intuitive so that the students can use it without substantial training.

Constraints

Main projects at HIG constitute a workload of 20 study points which, according to the policy for main projects, constitute about 400 hours or approximately 25 hours a week through one semester.

1.2 Extent of task

Task description

An application for creating stop-motion animations for use in primary and secondary schools. The students should be able to put together animations from still pictures on the hard-drive or by capturing pictures from a camera connected to the computer. It should also be possible to add sound effects, background music, subtitles and possibly other, more advanced, effects. When the students have made an animation they should be able to export it to different video formats such as mpeg or avi.

The application will be a part of the Skolelinux solution for schools and is primarly meant as a pedagogical tool in the education.

The platform

The target platform for this application is the Skolelinux system based on Debian Linux and KDE Desktop.

GUI

To build the graphical user interface to the application we have selected the QT library from the Norwegian corporation Troll tech. There are several reasons for this:

- It is widely used and recognized GUI-library which is used allot, both in Open Source projects and in commercial companies
- KDE, which is the desktop Skolelinux is based on, is built on QT and it is therefore a logical choice
- It is a class library which, in our opinion, is a big advantage

- Platform independence
- We feel it is exciting and look forward to working with it

Even though we have selected QT we can see that someone may wish to port the application to other GUI libraries in the future. We will therefore gather the GUI specific code and separate this from the rest of the program, so that it will be fairly easy to change it at a later time.

Other GUI libraries which have been considered is:

- GTK
- FLTK

Other libraries, frameworks and standards

In the pre-project phase we have identified some other libraries, frameworks and standards which we may need in the software development. These are mainly:

- STL
- SDL
- OpenGL
- GStreamer
- ffmpeg
- XML

Programing languages and development environment

The group consider the programing language C++ to be a reasonable choice because the selected GUI library is in this language and because we are comfortable with it. Other languages, especially C, will also be relevant to some parts of the development partly because gstreamer, which we will probably use to handle images, sound and video, is implemented in it and because the C++ bindings for gstreamer are incomplete as of now.

The development environment we have picked is Kdevelop. the reason for this is that Kdevelop is a powerful and helpful tool which is especially suited for the platform and the type of development we will be doing (open source).

Constraints

Since the project is fairly large and the student group only consist of two members, it isn't certain we will have time to implement all the functionality which may be wanted. We will therefore concentrate on the core functionality first such as importing and capturing pictures, putting together the animation and exporting to video. When this is done we will proceed to more advanced functionality such as sound and sound capturing from microphones and subtitles.

We will however not only consider the most wanted functionality when choosing what to develop first, but also account for technological risk and the architectural importance of the requirements. For more about this see Appendix **??**

1.3 Project organization

The two members on the project group will have different roles and areas of responsibility. Most of these will be distributed as we go, but some responsibilities which are already set are:

- Fredrik is responsible for the pre-project report
- Bjoern is responsible for the requirement specification

1.4 Planning and reporting

Choice of methodology

As our basic methodology we have chosen an approach based on evolutionary system development, but we choose to adapt this so that we gain more support than what is provided in the raw methodology. How we will adapt this methodology is explained in Appendix **??**.

We feel that this way of working will suit this project very well considering the fact that we are doing open source development. Our alternatives were mainly incremental development and eXtreme Programming. Incremental development were discarded because we didn't feel it was sensible to plan all the increments in this type of project up front and we didn't choose eXtreme Programming strictly because we lacked the on-site customer.

Our model, in contrast to the incremental model, allows us to plan the iterations just before they start which we feel adds to our flexibility.

Plan for status meetings and decision dates.

Formal status meetings will be held on the following dates: 25/02, 28/03 and 26/04 but more informal meetings with the project supervisor will be held every week to every second week. These meetings will be natural forums for making large decisions regarding the project.

Code convention

To make the source code as readable and maintainable as possible, both for us and for other who may later have an interest, we will follow the code convention which is specified in Appendix **??**, but this convention, in the same way as all project files, may be changed at our discretion.

1.5 Organization of quality assurance

Routines and rules withing the group

As mentioned earlier we are only two member and therefore we will take lightly on such rules and create them as we need them. However some routines are already created:

- As much as possible of the work should be done from the group room at A034
- If conflicts starts within the group we will first try to solve it with peaceful discussion. If this is insufficienc we will attempt to solve it by the tried-and-proven method of paper-rock-scissor. The next step will be to try and solve the problem through a decent snowballfight with street rules. As a last resort the project supervisor will be asked to mediate. This will, of course, be done under the watchful eye of tux

Revision management

In this project we will use CVS to handle both source code and project documentation. The alternatives are mainly manual version handling which is out of the question, Subversion and Bitkeeper. The main reason to why we have chosen CVS is that the customer have asked us to use this, since they use it extensively.

We are however used to CVS from earlier projects and would likely have picked this alternative anyway.

Quality handling in the source code

To ensure the quality of the final product the following methods and tools will be used:

- Docbook to generate documentation for the end user
- Doxygen to document the source code and program api
- Pair programming of especially difficult and/or critical parts of the code

Risk analysis

We have identified the following project risks:

The project dies with us. This is always a big risk with final engineering projects. We will try to avoid this by trying to get people interested in the application. If someone needs the application they will not want to see it die, and maybe consider contributing to keep it alive by patching it. The way we will try to spark an interest is by promoting the application as it matures to both the Skolelinux and to graphical communities.

Sickness This could be a huge problem for us since we are only two developers. We will try to minimize the impact of this risk by teaching each other about the things we are working on, so that one person don't sit on vital information. This way the other person can keep working efficiently if the other one get sick.

1.6 Development plan

As stated earlier we don't wish to plan all the entire project up front as it would have been done in a waterfall or incremental project. By not planning all the iterations early we gain more flexibility and will be more responsive to changes. The way we see it much of the benefit in iterative development is the ability to learn as we go, and use the experience from the N first iterations when planning iteration

N+1. This is the way it is done in most modern development methodologies such as for example RUP, XP and SCRUM.

We will therefore only give a coarse plan of the entire development here, and specify early iterations more detailed. We will specify the later iterations just before they comence as specified in Appendix **??**. We have however, as previously stated, added a list in Appendix **??** specifying the ranking of the requirements at this point. This list will however change as we go. An coarse grained overall plan of the project can be found in appendix **??**.