Pre-project report for the Stop-motion Animation Project

Fredrik Berg Kjoelstad Bjoern Erik Nilsen

12th May 2005

1 Goals and constraints

1.1 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.

1.2 Effect goal

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

1.3 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.

1.4 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.

2 Extent of task

2.1 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.

2.2 The platform

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

2.3 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

2.4 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

2.5 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).

2.6 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 B

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

4 Planning and reporting

4.1 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 A.

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.

4.2 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.

4.3 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 C, but this convention, in the same way as all project files, may be changed at our discretion.

5 Organization of quality assurance

5.1 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

5.2 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.

5.3 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

5.4 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.

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 A. We have however, as previously stated, added a list in Appendix B 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 B.

Appendixes

A The System development methodology

Our system development methodology is an adaption of evolutionary development with a more structured process.

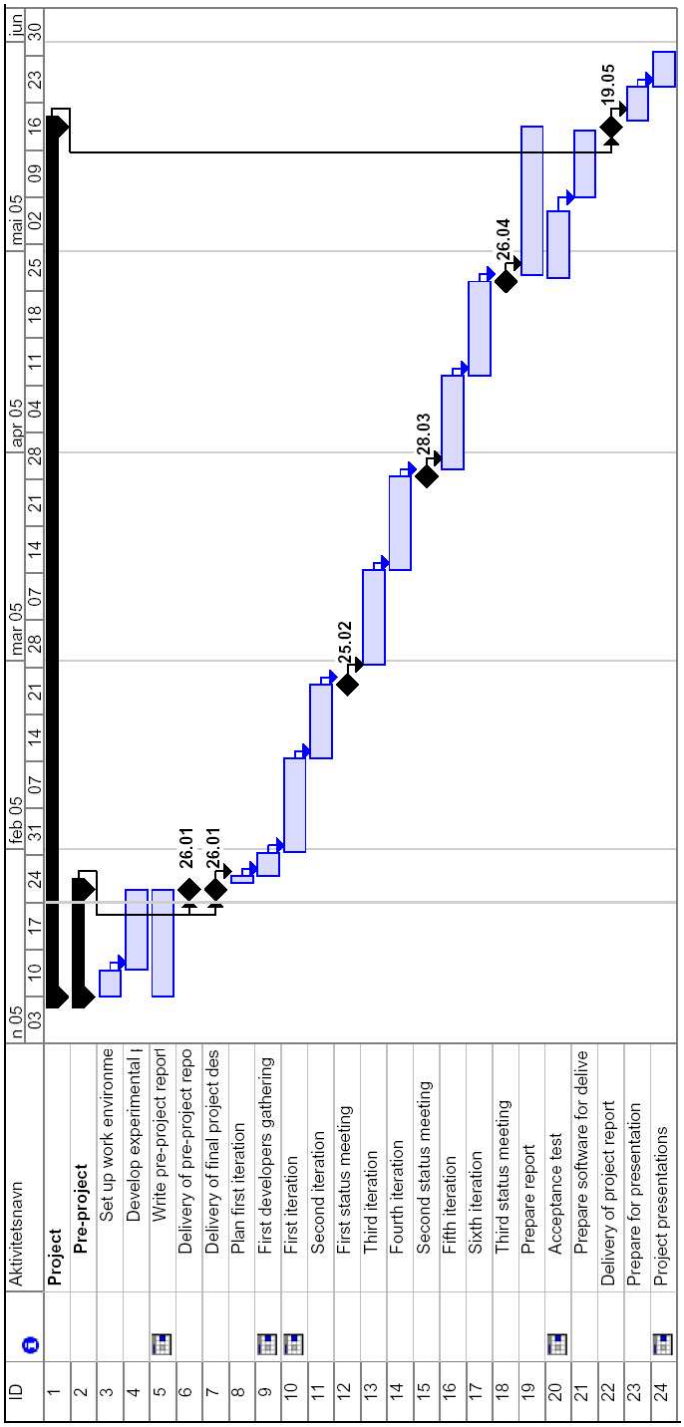
The workflow is divided in several two-week iterations which begins with a phase where the architectural integrity and new requirements are evaluated. After this a set of tasks for this iteration are selected. Such tasks may for example be new requirements to implement, refactoring of code or project tasks.

When a set of tasks are selected which are deemed sufficient for two weeks the development process will be freezed. New requirements which are proposed in this period are noted, but for the most part not evaluated or worked on until the next iteration. This is done so that the development can proceed without distractions, and we hope this will make us more efficient. This is however a general rule, not a law.

drik/skolelinux/www/info/studentgrupper/2005-hig-stopmotion/project_management/development_plan/methodology_dia

B Project plan

Following is a coarse plan for the entire project workflow.


ID
Aktivitetsnavn
n 05
feb 05
mar 05
apr 05
mai 05
jun
03 10 17 24 31 07 14 21 28 07 14 21 28 04 11 18 25 02 09 16 23 30
1 Project
2 Pre-project
3 Set up work environme
4 Develop experimental
5 Write pre-project report
6 Delivery of pre-project repo
7 Delivery of final project des
8 Plan first iteration
9 First developers gathering
10 First iteration
11 Second iteration
12 First status meeting
13 Third iteration
14 Fourth iteration
15 Second status meeting
16 Fifth iteration
17 Sixth iteration
18 Third status meeting
19 Prepare report
20 Acceptance test
21 Prepare software for delive
22 Delivery of project report
23 Prepare for presentation
24 Project presentations
26.01
26.01
25.02
28.03
26.04
19.05

C Requirements ranking:

Requirement	Type	AS	Risk	Criticality	Sum
Import picture	UC	2	3	3	19
Create movie	UC	3	3	2	18
Manage animation project	UC	3	0	3	15
Internationalization	Feat	3	0	3	15
Create frame	UC	3	0	3	15
Run animation	UC	2	2	2	14
Setup camera	UC	1	2	2	12
Run scene	UC	2	2	1	11
Import sound	UC	2	2	1	11
Easily ported gui	Feat	3	2	0	10
Undo/redo	UC	1	1	2	10
Logging	Feat	3	0	1	9
Manage scene	UC	2	1	1	9
Copy frame	UC	0	1	2	8
Add subtext	UC	1	2	0	6
Set speed	UC	0	3	0	6

Table 1: Table showing the requiremets and weights showing how much impact they have on the different important areas when ranking the requirements

* Depends on how this will be implemented. As an advanced slideshow or by using Create movie to create a movie which is the run.

	Weight	Range
Architectrually significant	2	0-3
Technical risk/complexity	2	0-3
Criticality/Early biz. value	3	0-3

Table 2: Table explaining the approximate weight of the specified area when prioritizing the requirements

D Code conventions

```
func(x, y) {
   ...
}
```

The indent size is one tab-stop for each level of indenting.

```
if (number == 2) {
   ...
}


if ( <komplexExpression> ) {
   ...
}
else {
   ...
}

while (n < 30) {
   fooBar(n++);
}
```

Note the convention on function naming and the presence of curly brackets even though there is only one line of code. else always start on it’s own line.

```
switch (currentToken)
{
   case 1:
   {
      ...
   break;
   }
   ...
}
```

```
class Foo
{
public:
   void aFunction();

protected:
   int someData;
   void anotherFunction();

private:
   char someMoreData;
   void yetAnotherFunction(data);
};


Foo *bar = new Foo();
```

- All variables are private or protected; private is preferred.
- All functions should be private or protected if possible.

```
//animation.cpp
#include "animation.h"

#include "foofile.h"
...

#include <barfile.h>
...
```

Other notes:

- Variables with a long lifetime and large scope such as private variables have long descriptive names such as motionPictureHashMap

- Variables with a shorter lifetime and smaller scope such as lokal or parameter variables have shorter names such as pictureMap
- Loop and short lived test variables have names consisting of one character such as i, j or n.
- Constants (const) are named in capital letters with an underscore between each word: IMPORTANT_CONSTANT.

E Requirement specification

E.1 Supplementary specification

Functionality

- All buttons, except from trivial ones(ok, cancel, ...), shall have a tooltip.

Usability

- The application should be colored and visually designed so that it has a simple feel.
- All aspects of the application must be internationalized so that it can easily be ported to multiple languages.

Reliability

- All errors and debug information should be logged through the same mechanism.

Performance

- The application should run smoothly on a 800MHz processor.

Supportability

- The application should be easy to port to new GUI-libraries and changing the GUI-library should not require changing many layers.

E.2 Usecases


Setup camera
Import picture
<<includes>>
Create frame
Copy frame
<<includes>>
Move frame
Import sound
User
Add subtext
Set speed
Run animation
Run scene
Create movie
Manage scene
Manage animation project

Name Import picture

Goal The goal is to import a picture into the selected frame (or frames).

Precondition If the user wants to import a picture from camera: The camera is set up for use in the application.

Postcondition The picture is added to the selected frame/frames.

Description A new frame is created *using "Create frame"* and a picture is imported into this frame, either from the storage medium, or from a camera.

Unresolved issues

- What if the user wants to import a picture into an existing frame?
- What about layers?

Technical implication

- Need library support for handling/interpreting picture formats and for recording them from the camera.

Main success scenario

1. The user request to add a picture to the animation
2. The system creates a new frame after the currently selected *using* the "Create frame" usecase
3. The system adds the picture to the new frame

Variations

1a. The user request to add a picture from a file on the harddrive

    1. The system ask where the picture file is located on the hard drive
    2. The user tells the system where the picture file is located on the harddrive

2a. The the picture file is not an supported file type

(a) The system informs the user that the file is not supported

(b) The system ask the user where another picture file is located on the harddrive.

1b. The user request to add a picture from a camera connected to the computer

1. The system takes a snapshot from the camera

Name Create movie

Goal The goal is to export the animation to a movie format such as vcd or mpeg.

Precondition There is an animation to export as a movie

Postcondition The movie file with the animation is created.

Description The program builds the movie file from the different pictures, sounds, subtexts, etc in the animation project, and stores it to a storage medium.

Unresolved issues

Technical implication

- Need library support for exporting to the different formats.

Main success scenarios

1. The user requests to create a new frame
2. The system creates a new frame behind the one currently selected

---

Name Add subtext

Goal The goal is to add a subtext to a frame or a (continous?) series of frames.

Precondition

Postcondition Subtext is added to the selected frames.

Description The user selects one or more frames and then specifies a subtext for them.

Unresolved issues

Technical implication

- Might want library support for raster or vector text. I think gstreamer has a plugin (demuxer) for this...

Name Create movie

Goal The goal is to export the animation to a movie format such as vcd or mpeg.

Precondition There is an animation to export as a movie

Postcondition The movie file with the animation is created.

Description The program builds the movie file from the different pictures, sounds, subtexts, etc in the animation project, and stores it to a storage medium.

Unresolved issues

Technical implication

- Need library support for exporting to the different formats.

Name Import sound

Goal The goal is to import a sound effects into the project from the selected scene and onward until the sound is played to completion.

Precondition The frame to link the sound to is selected.

Postcondition The sound is linked to the selected frame.

Description A sound is either imported from a sound file or recorded from a microphone. The sound is then added to a frame and starts playing when the movie reaches the frame.

Unresolved issues

- Different formats (MP3, WAW, etc)
- Special considerations for the background music or should this just be inserted as a long sound started from the first frame?

Technical implication

- Need libraries for interpreting the sound files, playing them, putting them into the video and for recording sound from the microphone.

Main success scenarios

1. The user request to add a sound to the selected frame
2. The system links the sound to the selected frame

Variations

1a. The user request to add a sound from a file on the storage medium

1. The system ask where the sound file is located on the storage medium
2. The user tells the system where the picture file is located on the storage medium

2a. The sound file is not an supported file type

(a) The system informs the user that the file is not supported

(b) The system ask the user where another sound file is located on the storage medium

1b. The user request to add a sound captured from a microphone connected to the computer

1. <The way this should be resolved has to be determined. Either use some other program, not allow it or build in some mechanism for capturing sound>

Name Run animation

Goal The goal is to show the animation with sound and other attributes to the user.

Precondition

Postcondition

Description Runs the animation by displaying one frame at a time together with sound and other attributes.

Unresolved issues

- How should the different component such as picture(s), sound file, etc be added together? On the fly or before the animation is run?

Technical implication

Name Run scene

Goal The goal is to show the selected scene as an continious animation with sound and other attributes to the user .

Precondition

Postcondition

Description Runs the scene as an animation by displaying the frames one at a time together with sound and other attributes.

Unresolved issues

- How should the different component such as picture(s), sound file, etc be added together? On the fly or before the animation is run?
- What about sounds which transcends scenes.

Technical implication

Name Copy frame

Goal To copy a frame with all it’s preferences.

Precondition A frame is selected.

Postcondition A new frame, identical, frame is created and put after the old frame.

Description The selected frame is copied and the copy is placed in the next frame slot. The old frame remains selected.

Unresolved issues

Technical implication

Name Setup camera

Goal To set up a camera to use for capturing pictures.

Precondition

Postcondition The camera is set up successfully and ready for use for image capturing and onionscinning.

Description The user adds the camera to the program and sets it up for use by the application.

Unresolved issues

Technical implication

- Library support for talking to different camera and webcam standards.