

Pre-project report for the DaisyPlayer Project

André Lindhjem, Kjetil Holien, Terje Risa and Øyvind Nerbråten

25.01.2006

Contents

1	Goals and constraints	3
1.1	Background	3
1.2	Effect goal	3
1.3	Result goal	3
1.4	Target group	3
1.5	Constraints	3
2	Extent of task	4
2.1	Task description	4
2.2	The platform	4
2.3	Libraries, frameworks and standards	4
2.4	Programming languages	6
2.4.1	Programming languages	6
2.4.2	Python	6
2.4.3	Ruby	6
2.4.4	C++	6
2.4.5	C	6
2.4.6	Availability of compilers and interpreters	6
2.5	Constraints	7
3	Project organization	7
3.1	Principal	7
3.2	Group	7
3.3	Roles and responsibilities	7
4	Planning and reporting	8
4.1	Choice of methodology	8
4.2	Plan for status meeting and decision dates	8
4.3	Code convention	8
5	Organization of quality assurance	8
5.1	Revision management	8
5.2	Quality handling	9
5.3	Risk analysis	9
6	Development plan	9
A	Risk analysis ranking	11

1 Goals and constraints

1.1 Background

Skolelinux is a Custom Debian Distribution that is customized for schools with focus on being easy to install and maintain. Skolelinux tries to give schools and students a good and free alternative to proprietary software. This is used in many schools already and the goal is to cover as many schools as possible.

In order to achieve this goal Skolelinux need software to cover the different needs the student have in their education. One such program, after a request of a Norwegian teacher, is an application able to play Daisy Digital Talking Books (DTB). Daisy DTB is a multimedia representation of a print publication. Linux lacks a good and free Daisy player at this point.

1.2 Effect goal

The effect goal is to start the development on a free Daisy player which, in an easy way, helps students play Daisy-books using Skolelinux or other Linux-distributions.

1.3 Result goal

Result goals are to start an Open Source project and to create an engine for playing Daisy DTBs with an API and a simple user interface. It must be able to play at least one audio format, and it should be easy for others to program against the API.

1.4 Target group

The main target group for this project is the students and other users which need a free Daisy player for Linux. But we must also have in mind that the project are going to be evaluated by the school.

1.5 Constraints

This project must be Open Source and licenced according to The Debian Free Software Guidelines (DFSG).¹ Software created must run on “Skolelinux” and it’s a goal to keep the software dependencies to a minimum.

¹DFSG http://www.debian.org/social_contract.html

2 Extent of task

2.1 Task description

The task is to create a software Daisy player for Linux. This software must follow the Daisy\NISO standard ². The program should be able to play the audio and display synchronized text. Navigating and searching through the books will also be supported.

A simple UI to the daisy engine will be made. A more advanced GUI have been requested, but has a low priority. The creation of a GUI depends upon the Daisyplayer engine and API and are limited by the projects time resources.

Main functionality:

- Make an engine able for audio playback.
- Display the synchronized text.
- Make a simple user interface which makes it possible for user interaction.
- Making a navigating- and searching functionality.
- The project should be easily extend able.

It's very important that we try to separate the engine from the user interface. This is done because it should be a simple job to change the user interface at a later time if wanted.

The principal may come with suggestions and more functionality on the way, but at this point the above points are our goals.

2.2 The platform

The Skolelinux system is based on Debian GNU/Linux and KDE Desktop and is the target platform for this application. We will try to make it portable to other platforms, but this is not a priority.

Audio output to aRts is a minimum requirement, but a goal is to support the other common methods such as als, oss, esd too.

2.3 Libraries, frameworks and standards

This far we have identified the following libraries, frameworks and standards which we may need in the development of this application.

²Daisy/NISO Standard <http://www.daisy.org/z3986/>

Audio

- Audio formats (mp3, mp4-aac, wav)
- Audio back ends (aRts, esd, oss, alsa)
- MAD (MPEG Audio Decoder)
- Initial research suggests that the libao library is a good choice for communicating with audio back ends. libao claims to support the back ends mentioned above.

Graphics

- JPEG
- PNG
- SVG

GUI

- GTK+
- QT

XML

- SMIL
- XHTML

IPC

- DBUS
- Sockets
- Pipes

Other

- Unicode
- Festival

2.4 Programming languages

2.4.1 Programming languages

We have evaluated a few programming languages to come up with the best choice in relation to the project. This project will use a lot of existing technologies, so the availability of libraries are important (we don't want to re-invent audio- or XML libraries). The project will also be split up into multiple strongly separated modules. As long as the modules supports the same IPC, there are possible to use different programming languages for the individual modules.

2.4.2 Python

Python is an object-oriented language with good support for technologies we are likely to use. Support exists for XML, Unicode, wrappers to audio output-libraries, widget toolkits etc.

2.4.3 Ruby

Ruby lacks proper support for Unicode, but has partial support for UTF-8. We must look into this in the event that we use Ruby for components that needs to deal with Unicode.

2.4.4 C++

C++ is a powerful language, but has some issues as we see it. The size of variables are not fixed, but depends upon the HW-platform and compiler. There have also been expressed worries that we might not get the compatibility and back wards-compatibility we need because of variation between compilers, HW-platforms and libraries.

2.4.5 C

C is a powerful language, but it does not fit for this particular project as we see it. It is not object-oriented and, even though the technologies we are going to use are supported through external libraries, we think there are better and easier alternatives which will allow us to implement our program faster.

There is a slight possibility we will have to review or modify existing code in libraries etc. These are likely to be written in C, so we must be ready to face the C programming language at some point through our project. But, as a primary programming language, C is not what this project requires.

2.4.6 Availability of compilers and interpreters

Compilers for C/C++, Java and more are shipped with Skolelinux.

An interpreter for Python are shipped with Skolelinux as default, while the Ruby interpreter needs to be installed as an extra. ³

³<http://www.skolelinux.org/portal/faq/general/software>

2.5 Constraints

A goal is to keep the software dependencies to a minimum, so it can be easily installed on a typical workstation. Because of time limitations we focus on the player engine and at least one UI. A future extension is to implement a GUI, but this is not a priority until the engine is complete.

3 Project organization

3.1 Principal

Skolelinux. Contact: Herman Robak.

3.2 Group

The project group consists of:

André Lindhjem, 03HBINDP HiG
Kjetil Holien, 03HBINDP HiG
Terje Risa, 03HBINDD HiG
Øyvind Nerbråten, 03HBINDP HiG

3.3 Roles and responsibilities

The four members on this group will have different roles and responsibilities. Most of this will be distributed as the need occur during the project, but some roles have already been assigned.

- Our project leader will be Øyvind Nerbråten, he will be responsible to make sure that all of us works in the terms we have agreed.
- Each one of the members are responsible to keep a log about the work they do and we will try to work about 25 to 30 hours a week. When we will work will mostly depend on the others lectures we got to attend to, but try to work at the same time of the day since we then can discuss problems and solutions that may occur.
- If a member don't work sufficient the group leader will talk to him and give him a warning, if that doesn't help, the teaching supervisor will be contacted.

<http://developer.skolelinux.no/cgi-bin/viewcvs.cgi/~checkout~/skolelinux/src/task-skolelinux/pkgdeblast.txt>

4 Planning and reporting

4.1 Choice of methodology

The group have decided to use an mixture of evolutionary system development and Open Source development. Since the project is an Open Source project we felt it natural too follow the Open Source development standard, with a few minor changes. We will try too release a new version every two weeks, giving the project some time boundaries and certain goals to work against. All our work will be on the Internet in a revision management system, giving users the option of downloading the work as it's develops, meaning that they don't have to wait for the new version to be released to be updated. We will also try to implement user suggestions and establish an user group that hopefully will find this project interesting, in true Open Source style.

The only other development choices we truly consider where incremental development or a more pure evolutionary development. Since we didn't feel that either one of the development methods mention before suited our needs, we decided to use an adaptation of the evolutionary model.

4.2 Plan for status meeting and decision dates

The formal status meetings will be held on these dates: 20/02-06, 20/03-06 and 24/04-06. But we will have more informal meeting with the supervisor more often. The status meetings will be a natural time to decide large decision which regards the projects outcome.

4.3 Code convention

We will try to keep the source code as readable and structured as possible. This is done to make it easier for both us, and other readers, to understand and maintain the code. We will agree on code conventions for the languages we use. Since we are planning to take advantage of existing code, we must be flexible on programming languages and code conventions.

5 Organization of quality assurance

5.1 Revision management

In this project we will be using Subversion to handle our version control of all our work. This includes both the source code and the project documentation. Because all Skolelinux projects are stored in a shared repository, the choice of manual version handling was eliminated, not that manual version handling was an eligible alternative either way. Since Skolelinux just started using Subversion instead of CVS, the choice was already made for us.

5.2 Quality handling

We must be dynamic and prepared to receive and act on user input to ensure quality. As most Open Source projects we are dependant upon user feedback to do testing maintain code quality.

The creation of an API for developers and a good users manual are a priority. Presentable versions of important documents created throughout the project are automatically generated within one hour and linked to from the project website.

5.3 Risk analysis

We have identified the following risks:

User feedback: Lack of feedback from the user community will make it difficult to develop software for people with special needs. To avoid this we must reach out to the open source community and advertise for the project in the right forums and mailing lists.

Future maintenance: It is important for this project survival to get users and other developers involved. It is imperative for the project to be active to maintain the user-mass. To do this, we must advertise the project.

Sickness: This is a common risk with software development projects. In our case this is not a very big risk, since we are four members with almost equally knowledge. And to prevent any serious problems that sickness may cause, we will keep each other updated with how things work and what we do.

Principal or target group: The risk of the principal or target group looses interest are considered to be minimal. There is a genuine need for a Daisyplayer for the Linux platform, and (as we are an open source project) we are not dependant upon the principal to the same degree as “traditional” projects are.

Violations of patents and trademarks: This could be a serious risk if we are not careful and thorough enough in our search for code we can use. In appendix A we have ranked the risks.

6 Development plan

As already mentioned under the choice of methodology we are working in a mixture of evolutionary system development and Open Source development. That means that we are not able to plan the whole project very specific, like we would have done with e.g. a waterfall model. Because of the fact that we must be prepared to accept and act on code-contributions, bug-reports and fixes, we must be flexible during the development. And since we are planning to release a version every other week, it means that we work in cycles. In each cycle, we

add more functionality. Each cycle consist of planning, coding and testing. A coarse grain sketch of our Gantt chart can be found in appendix B.

A Risk analysis ranking

Nr	Risk	Probability	Consequence	Measure
R1	User feedback	High	Medium	If we don't get feedback we want and hope for we just have to advertise more and call people we know would be interested.
R2	Future maintenance	High	Medium	Advertise more for the project and hopefully get a skilled successor.
R3	Sickness	Medium	Low	We will keep each other updated with how things work and what we do.
R4	Principal or target group	Medium	Low	In order to keep the target group interested we have to come with releases often and advertise.
R5	Violations of patents and trademarks	Low	High	Be careful and thorough in our search for code.

B Gantt chart



