Speech/music splitter

Case study/prototype report

Investing in the future by working together for a sustainable and competitive region

Abstract

During the autumn of 2010 a team of six students from Karlstad University, Sweden, worked on an assignment from Hanze University in Groningen, Holland. This report is their description of the project work which consisted of development of a speech/music separator to be used by the student radio organization within Hanze University.

DVGC06

HT 2010 ▪ FINAL REPORT

BEATRIZ SÁNCHEZ

ALI YANAR

DANIEL MESTER PIRTTIJÄRVI

SEBASTIAN THERNSTRÖM

HAMPUS SKYSTEDT

AGNI RIZK

Table of content

1. Introduction 1
2. Technical 2
2.1 Description 2
2.2 Theory 3
2.2.1 Digital audio 3
2.2.2 Stereo audio 4
2.3 Approaches 4
2.3.1 Amplitude 4
2.3.2 Comparison of channels 4
2.4 Workflow 6
2.5 Class implementation 6
2.5.1 Interval 7
2.5.2 Analyser 7
2.5.3 Splitter 8
2.5.4 ProcessingQueue 8
3. Non-technical part 9
3.1 Customer interaction 9
3.1.1 Relation to customer 9
3.1.2 Customer value 9
3.2 Project work 10
3.2.1 Method and process 10
3.2.2 Group dynamics 11
3.3 Experience 11
3.3.1 What knowledge have you used, relation to theory 11
3.3.2 What would you do differently if you had to do it again 11
4. References 12
5. Appendix A: Source code 12
5.1 Analyser.cs 12
5.2 Splitter.cs 17
5.3 ProcessingQueue.cs 22
5.4 Interval.cs 26
5.5 MainForm.cs 27

1. Introduction

This report will describe our project work with a branch of the school Hanze University in Groningen, Holland, which is responsible for a radio program they run every week with information and entertainment for, and by, their foreign students. They run the radio program on a web portal called HappyHourFM, but the interest for the program has made it outgrow the built-in functionality of the portal and they came to us with a plan to integrate their radio station in to a new web site based on WordPress, with extended functionality for news reports, searchable archive and stored podcasts.

They had an idea where the radio shows would be stored and tagged on the site with keywords from what the hosts have been talking about. They would be searchable and playable from an archive on the site so that students could get old information they might be interested in. But to do this without breaking any rules of copyright protection to the music played during the shows, only the parts of it during which the hosts are talking can be stored. They wanted this to be done automatically, which meant we had to develop a program that could do just that, without too much user interaction and involvement.

2. Technical

2.1 Description

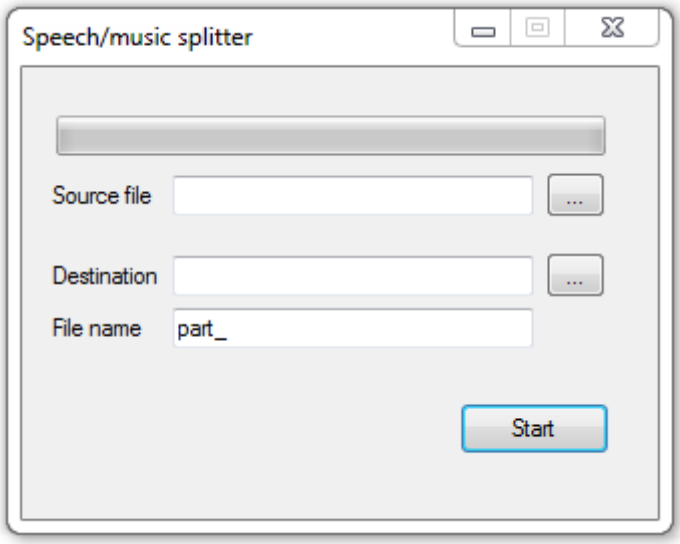

Figure 1. This is what the GUI of the application looks like.

The program consists of a user interface, where the user can fill in the directory location of the desired source file and the destination, as well as file name, of the produced files. The only error prevention we have here is that if one of the three text fields is empty the user is asked to fill them in correctly with a message box.

Once the fields are filled in, the user press a button labeled "Start" to begin the editing procedure. The application starts with analyzing the file, finding out during which time intervals there is speech and when there is music. Once the analyzer is done the splitter takes over and splits the audio file into several separated audio files, which are supposed to only contain speech, and the music is discarded.

During the entire editing procedure, both the analyzer and the splitter, the user can see the total progress in the form of a progress bar and a label showing the estimated time remaining. When working with an entire radio show podcast this can be quite useful since the entire process will take up to 10 minutes or even more.

While the application is working, the label of the "Start" button will change to "Cancel". And instead of starting the process anew, pressing the button will immediately interrupt the analyzer and splitter and return the interface to its original state.

If the process is allowed to finish without interruption, the remaining-time label will change to "Done!" and a button with the label "Exit" will appear, the "Cancel"-button will also change back to "Start". From here, the user can either quit the application by pressing "Exit" or start the process from the beginning, preferably with another source file and desired name of the produced files.

2.2 Theory

The main task in this project was to extract the speech parts from a recorded radio show. During the preparation of the project multiple approaches were tested and evaluated. These will be described in section 2.3. Before describing the approaches we need to find out how audio is represented binary in a computer.

2.2.1 Digital audio

In reality audio is a wave that consists of compressions that is propagated through the air. To be able to process audio in a computer this wave must be converted to a digital sequence. The conversion from wave in the air to a digital sound file is performed in two steps:

1. CAPTURE WAVE. In the first step the wave is captured by using a microphone. The microphone basically converts the mechanical waves in the air into electric signals.

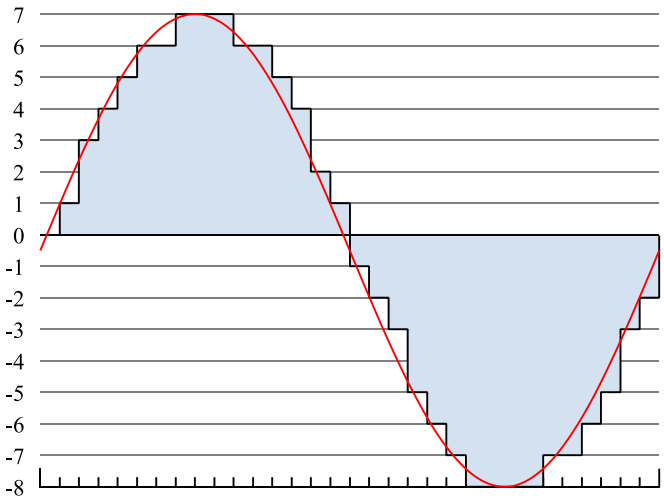

Figure 2. This diagram shows an analog electric wave (in red) and a 4 bit signed PCM wave (in gray). Source: Wikipedia.

2. CONVERT TO DIGITAL REPRESENTATION. The electric signal is not yet ready to be processed by a computer, since it is analog. Therefore an ADC, analog-to-digital converter, is needed. The most common way of representing audio in computers is by using PCM, pulse-code modulation.

When PCM is used, the ADC will measure the voltage of the electric signal a constant number of times every second. The number of times per second is given by the *sample rate*. A common sample rate is 44,100 Hz, which means 44,100 measures are done every second. The measures are then converted to numbers in digital form (e.g. signed 16 bit integers). Each measure is called a *sample*. (Pulse-code modulation, 2011)

You can see the difference between the analog electric wave and the digitalized content in figure 2. As you can see some information is lost in the conversion, but the losses are not noticeable.

2.2.2 Stereo audio

Today a lot of audio transmissions have more than one audio channel. Even in FM radio transmissions stereo audio is used. To be able to store multiple channels in an audio file, there are mainly two approaches available: time-division multiplexing, TDM, and frequency-division multiplexing, FDM, where the former is commonly used in digital contexts. In wave-files the channels are usually interleaved in the following way:

LEFT CHANNEL SAMPLE, RIGHT CHANNEL SAMPLE, LEFT CHANNEL SAMPLE, …

2.3 Approaches

During the project mainly two approaches to detect speech parts were tested and evaluated. These will be presented in this section. Other approaches that were considered was to detect the beat in the audio, and to detect differences in frequency distribution, but these were never implemented and thus not fully evaluated.

2.3.1 Amplitude

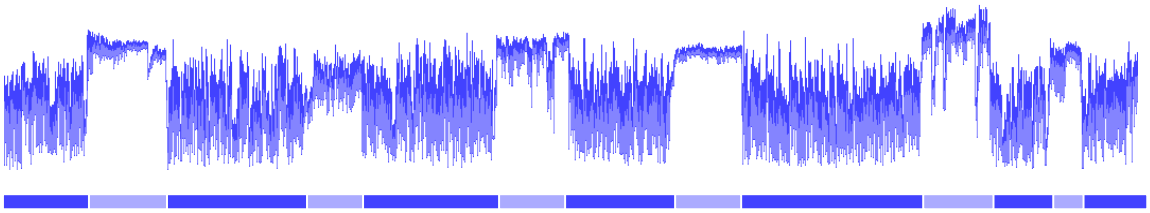

Figure 3. This is the EWMA value of the amplitude through a whole recorded radio show. The darker part indicates speech parts.

CHARACTERISTICS: When music is played the amplitude of the recording will be higher than during the speech part. The variation of the amplitude will also be higher during speech, since speech contains silent parts.

APPROACH: Calculate an exponentially weighted moving average, EWMA, on the samples in the sound file. An example of the resulting EWMA is displayed in figure 3. As you can see a clear difference between the speech and music parts is visible on the EWMA graph. However, we abandoned this approach in favor of the approach that will be described next.

2.3.2 Comparison of channels

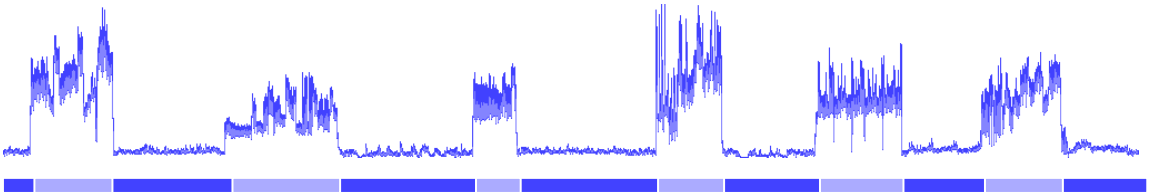

Figure 4. This is the EMWA value when applied on the difference between the two channels. The darker parts are speech.

CHARACTERISTICS: In radio broadcasts music is usually played in stereo, while the speech is recorded using a single microphone, which means the speech will be in mono. The speech may be extracted by looking for parts where the difference between the amplitude in the channels is low.

APPROACH: To detect the speech parts the following is done:

1. Select two corresponding samples from the left and right channel. Let a_L be the amplitude of the sample in the left channel and a_R the amplitude of the sample in the right channel.
2. Calculate the total amplitude $A = |a_L| + |a_R|$. If A is less than a constant threshold T_a, the sample is skipped, since silent part affects the EWMA described in step 4 negatively.
3. Calculate the difference between the channels as $\Delta a_i = |a_L - a_R|$
4. Calculate the EWMA on Δa as $M_i = (1 - \alpha) \cdot M_{i-1} + \alpha \cdot \Delta a_i \quad (M_1 = 0)$
5. If M_i is less than a constant threshold T_d, the sample is considered speech, otherwise it's considered music.
6. Two fences are used to prevent spikes that is incorrectly considered music or vice versa:
    a. When detecting a music part that is shorter than a constant threshold T_m, the music part is not considered music, but included in the adjacent speech parts. Vice versa; when a speech part shorter than the constant threshold T_s is detected, it is ignored and included in the adjacent music parts.
    b. All speech parts shorter than a constant threshold T_r are removed.

A lot of time has been spent on calibrating the thresholds. The final thresholds used in the project are the following:

$T_a = 4.0\%$

$T_d = 0.8\%$

$\alpha = 0.01\%$

$T_m = 5$ seconds

$T_s = 2$ seconds

$T_r = 20$ seconds

2.4 Workflow

This is the main tasks performed by the application:

1. The user specifies the input file and the directory where the output files will be added.
2. The **ANALYSIS PASS** will execute the algorithm described in the previous section to determine the intervals that contain speech in the input file.
3. During the **EXTRACTION PASS**, each of these intervals will be extracted to separate files.

2.5 Class implementation

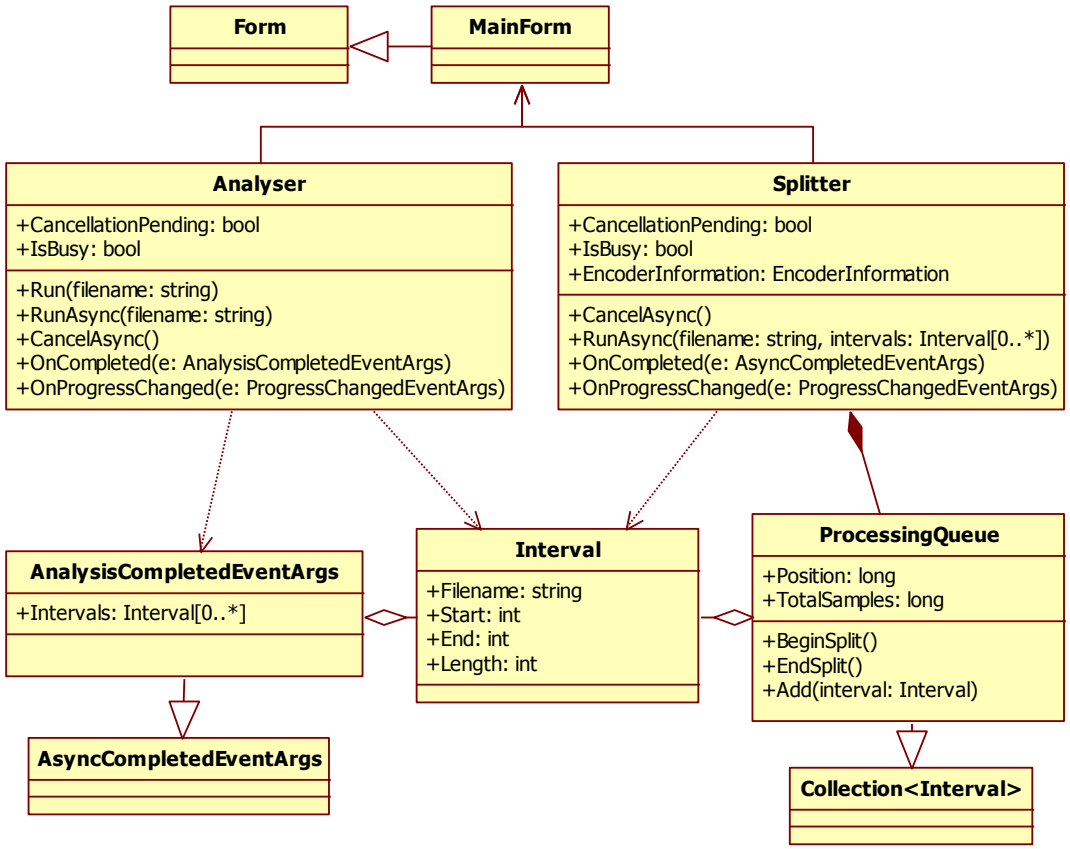

Figure 5. Overview class diagram of the classes used in the application.

In this section the main classes are described. In addition to the classes written by us, the application uses FFmpeg to decode and encode the audio files. Since FFmpeg is written in an unmanaged language, and the application written by us is written in C#, a managed language, all calls to FFmpeg go through a wrapper library called FFmpegSharp. The FFmpegSharp library was unfortunately pretty unstable when we started working with it, so the first weeks were spent on solving problems in FFmpegSharp.

2.5.1 Interval

The interval class represents a speech interval and holds information about the beginning and end of the interval, as well as the filename of the target file to which the interval will be saved.

2.5.2 Analyser

The Analyser class is solely responsible for the analysis pass. The analysis is performed in two steps:

1. Decode a pair of samples using FFmpegSharp. The first sample is the sample from the left channel, and the second sample is the one from the right channel.

2. Use the algorithm described in section 2.3 to decide whether the samples should be considered as speech or music. The algorithm is however implemented in a slightly different fashion than previously described. One difference is that the removal of speech intervals that are too short is performed in an own phase than the other actions.

The GUI lets the user specify any MP3, WAVE and WMA files as input files, but any file type supported by FFmpeg should work. However other formats have not been tested.

Since the input files that are supposed to be processed by the application are large, usually over an hour, the analysis pass will take a while to complete. If the analysis would be executed on the GUI thread, it would mean that the GUI would be blocked until the analysis is completed. The user will experience that the application has "frozen", and might kill the application before it has finished.

To avoid this, the class implements the event-based asynchronous design pattern (Microsoft), which is used e.g. by the BackgroundWorker class in .NET Framework. The class has the two methods RunAsync and CancelAsync. When RunAsync is called, the method will create a new thread and execute the operations described above in this new thread. The method will thus return immediately, and the user will not experience that the application has "frozen". The process can be cancelled at any time by calling the CancelAsync method.

The Analyser class also has two events: ProgressChanged and Completed. The ProgressChanged is raised frequently during the process to report the progress of the process. The progress is reported in percentages. The Completed event is raised in three cases:

- When the process has been successfully finished
- When an error has occurred during the analysis.
- When the process is cancelled.

Which case is relevant is specified by the event arguments.

The analyzer will only use one CPU core.

2.5.3 Splitter

This class is responsible for the extraction pass and also implements the event-based asynchronous design pattern. Since the splitter will process each interval returned by the analysis independently, the processing of intervals are divided into several threads. The number of threads is decided by the number of CPU cores available. The operating system will recognize that each thread is CPU intensive and put one thread per CPU core.

The process in each thread is actually not performed by the Splitter class, but rather delegated to the class ProcessingQueue described below. Upon a call to the RunAsync method, the splitter will create the necessary number of ProcessingQueues and distribute the intervals on the available queues so that the queue workload is approximately even.

Longest job first is the approach used for load balancing. This is how the intervals from a set of unassigned intervals, **S**, to a set of queues, **Q**, are distributed:

1. Choose the longest interval **I** from **S**.
2. Assign **I** to the processing queue in **Q** that has the lightest workload.
3. Remove **I** from **S**.
4. Repeat until **S** is empty.

The progress at any time is determined as the progress of the queue in Q that has most work left to do relative to the total amount of work in that queue. A timer is used to report the progress periodically by raising the ProgressChanged event.

2.5.4 ProcessingQueue

The ProcessingQueue is a class that is only used by the Splitter class and is responsible for the actual extraction of speech intervals to separate files. The extraction involves these operations:

1. Choose an Interval from the queue.
2. Open the input file and seek to the beginning of the interval.
3. Decode a chunk of samples from the input file and encode it to an output file. Repeat until the whole interval has been encoded to an output file.
4. Continue with next interval.

The encoding of the output files are determined by the EncodingInformation property of the splitter instance that hosts the processing queue. The GUI currently lacks options for selecting the output encoding, which defaults to 128 kbps CBR MP3, 44.1 kHz, 2 channels.

3. Non-technical part

3.1 Customer interaction

Here's some information about how we experienced the customer interaction, with a customer in Holland.

3.1.1 Relation to customer

When we started this course we got a project called "Digital Radio Station" and our contact person was Frank Willems. Our initial specification wasn't very detailed so we e-mailed Frank for further instructions. Unfortunately Frank wasn't so closely connected to the Radio-project and couldn't give us any understandable insight. He eventually got us in touch with Roel Hoving who had a better idea of what needed to be done and his English was much better. Despite this, Roel had a lot of small, unfinished ideas of what they wanted us to do and since we only had contact by e-mail it took a lot of time to sort out the most important tasks, and to fully understand them. This was the closest we came to a product backlog.

We eventually decided to start with software. We did separation of speech/music to remove copyrighted material. While we were doing this, trying out different algorithms etc., we didn't need Roel as much. Even though we didn't need his "help" during this time, we sat up a Skype meeting to get to know each other, discuss our ideas and further assignments.

If we had any questions, we would e-mail him, sometimes we got an answer in 24 hours and sometimes noting at all. We spent a lot of time waiting for answers. Obviously we worked on other parts (like GUI) but this was quite frustrating, so we eventually started to make our own decisions on witch priority different tasks would have.

When the project started to draw to its end, we asked Roel how he wanted the program, if he wanted a Skype meeting, where we could demo the program for him or if he just wants the program e-mailed to him with description on how to use it. Since he hasn't answered on the last two e-mails, we decided to e-mail him the program with instructions.

3.1.2 Customer value

From the beginning, the customer had a lot of ideas and suggestions for features we could develop, but didn't really have any idea of how we could do so. The one we started with, to separate speech from music parts of a recorded podcast didn't seem to be very hard to do but proved to be a lot more difficult than we anticipated. Mostly because the task the software was supposed to automate was very precision sensitive and we had to make sure to minimize the amount of error as close to none as we could.

This, combined with the amount of time spent communicating with the costumer at the beginning trying to figure out where to start and the fact that we had a limited amount of time to spend on the project made us decide to focus on this one feature and try to make our software solution as useful and user friendly as possible. In the end we didn't

accomplish as many tasks as we thought we would when we started because the one we did finish took a lot more work and time than we thought it would.

Since the customer didn't really have an idea of how the software they requested would work they didn't have any real expectations on how far we would get either. When we demonstrated what we had done to them they were surprised by the complexity and impressed with what we had done.

The end result was a solid and useful application that will come to good use for the customer as they continue to improve the radio program and web site in the future. They didn't have to spend much time on this project, apart from a few e-mails and a couple of Skype-meetings we have been working on our own. So even if there is still a lot left to be done for the customer, the value of what we have been able to produce in the time we had is quite good.

3.2 Project work

3.2.1 Method and process

We strictly worked against the Scrum technique. Joking aside, we tried to work with the Scrum technique. Daniel and Hampus were new to Scrum so they read "Scrum and XP from the Trenches" (the PDF from the course homepage). Agni, Ali, Sebastian and Beatriz had taken the Software Engineering course so we had some Scrum-experience. We discussed Scrum with Daniel and Hampus, everyone was excited about working with Scrum, but since we had some problem speaking with our product owner we weren't able to do a proper product backlog. So we abandoned the Scrum technique.

So what we did was to write down what needed to be done, what we needed to ask Roel etc. on a whiteboard. We divided up in smaller groups (or pairs) and did the tasks we liked doing the most. Then we checked the "tasks" when we did them, or got an answer for them. At the end of the day we had a mini retrospective, where we discussed the tasks we had left to do for the next day or the next occasion.

This is an example of what the whiteboard looked like in the beginning of the project, when we had a lot of thoughts and ideas.

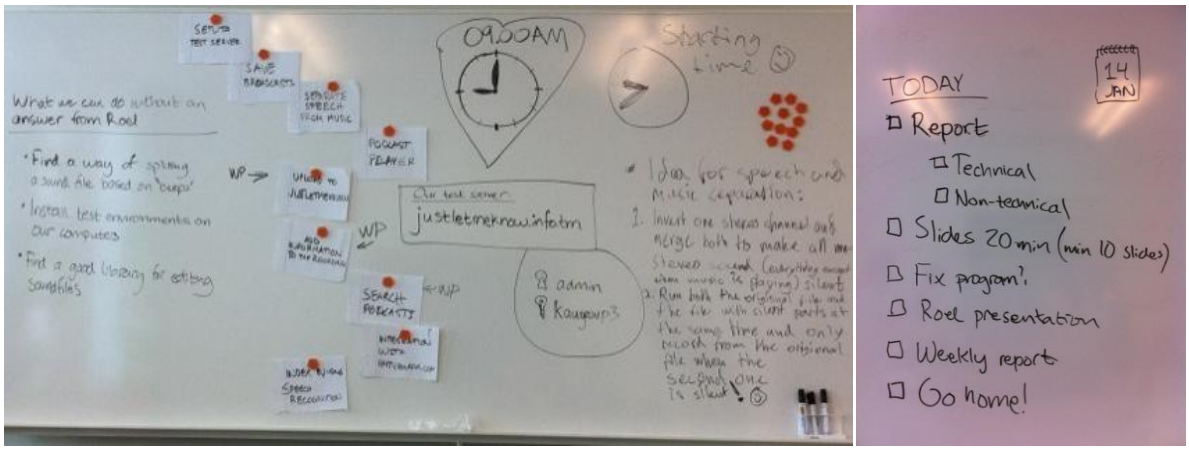

Figure 6. This is an example of what the whiteboard it looked like in the end of the project.

3.2.2 Group dynamics

The group's atmosphere was very good, we had a lot of fun despite our differences. We have people who really love and are good at programming, problem solvers, social/outgoing persons and people from different backgrounds. It was a nice mix, but sometimes we had problem speaking English with each other. This is a problem because Beatriz is an exchange student from Spain, and she doesn't speak Swedish. We discussed why this happened and we think that the underlying reason is that sometimes we simply forgot to speak English and sometimes it "flows" much easier and goes faster to explain something in Swedish. Swedish is after all, the remaining group member's mother language.

Another problem was that when group members were absence they did not always tell the rest, which created unnecessary annoyance. This could have been avoided if we had a communication in the group.

Sometimes we felt like the knowledge in the group weren't evenly divided, we think this is because when we got better contact with the customer we didn't have much time left to work on the project so we had to narrow it down. And when we did that, it became a very small project.

3.3 Experience

3.3.1 What knowledge have you used, relation to theory

When deciding how to edit the audio files we had to put our knowledge about digital sound representation to use. We knew about how a sound wave is split in to samples and how the samples are represented by bit values. This helped us to figure out how to compare the values of the different samples in the different stereo channels to determine whether it's music or speech. Some of us had taken a course in media compression and knew how audio compression works which helped us a lot.

We wrote our application in the programming language C# which we were all familiar with. Some of us had a lot of experience with C# in visual studio which came to good use when we started to work with external libraries and frameworks for the encoding and decoding of audio.

3.3.2 What would you do differently if you had to do it again

We had a hard time to decide how to begin working on the project when we first had the dialogue with the customer, it would've been very time conserving if we would've just focused on one part right away. If we were to get the specification now we would've tried to figure out one end to start with, in this case in the studio where the radio shows are recorded. What is the first task in the process from the studio to the web site that needs new software? In this case this would've been the software to edit the recorded podcasts, which we eventually started with.

In the beginning we asked both Frank and Roel what they wanted and to try to priorities the tasks they wanted to be done, but they didn't do that. So it was hard for us to decide that for them. If we had to do it again, we would have pressure the customer for a better project specification with priorities.