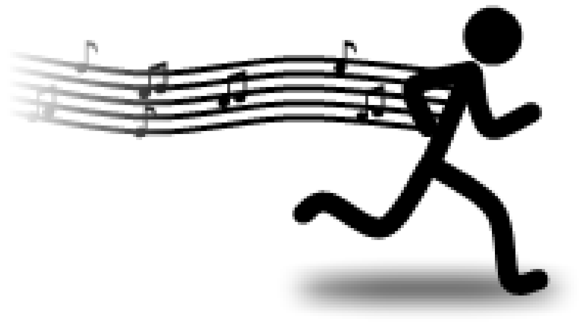
Tune Trainer



P3-project

by Computer Science group ds301e13

Aalborg University
The Faculty of Engineering and Science
Institute of Computer Science
P3-project - 3. semester
Supervisor: Fulvio Lizano Madriz
December 18, 2013

AALBORG UNIVERSITY
STUDENT REPORT

Aalborg University
The Faculty of Engineering and Science
Institute of Computer Science
Selma Lagerlöfs Vej 300
Telephone +45 99 40 99 40
Fax +45 99 40 97 98
http://www.cs.aau.dk

Title:
Tune Trainer

Subject:
How to improve exercise with music

Project period:
P3, 5. sep - 20. dec 2013

Project group:
ds301e13

Group members:
Anders Hermansen
Joachim Klokkervoll
Lynge Poulsgaard
Mike Pedersen
Samuel Nygaard Pedersen

Supervisor:
Fulvio Lizano Madriz

Copies: 7

Page numbers: 108

Source code: Enclosed on CD

CD's attached: 1

Finished: December 18, 2013

Synopsis:

This report describes how the mobile application Tune Trainer is developed. The research on exercise combined with music indicates that music have a positive effect on the exerciser, which was confirmed by an expert on the subject.
Tune Trainer is a simple piece of software that provides the user with the opportunity to make an exercise plan and hereafter play music to the current pace defined by the exercise plan.
The program was developed in C#, based on a construction method, where the system has been analyzed, designed, implemented and tested.
In order to develop a user friendly system, the program have been tested by ten individual users with no prior experience with the program.
Finally this report contains an academic report, which among other include an analysis of the k-means algorithm.

Aalborg, December 18, 2013

Anders Hermansen
student ID: 20124269
mail: aherm12@student.aau.dk

Joachim Klokkervoll
student ID: 20124277
mail: jklokk12@student.aau.dk

Lynge Poulsgaard
student ID: 20124272
mail: lkpo12@student.aau.dk

Mike Pedersen
student ID: 20124283
mail: mipede12@student.aau.dk

Samuel Nygaard Pedersen
student ID: 20124278
mail: snpe12@student.aau.dk

Preface

This report is written by 3rd semester Computer Science students at Aalborg University, fall semester 2013. Due to the technical level in this report, a basic knowledge of computer science is necessary to read and understand this report.

The report consists of two main elements, a Development component and an Academic part. The Development component is divided into three parts ("Analysis", "Design" and finally "Test and Reflection"). This is followed by the Academic part containing the process of the project and an analysis of the K-Means algorithm used in the project. At the end there will be an appendix containing additional material to the Development component and the Academic part.

Each chapter starts with a brief introduction describing the upcoming content.

References and citations noted with number notation, e.g. [1] which refers to the source in the bibliography with the same number. The bibliography is sorted alphabetically. When the word "we" is written it refers to the project team members.

We would like to thank the persons who participated in our test of the prototype as well as the persons we interviewed, Marina Aagaard and Susanne Stamhus. A special thanks go to our group supervisor Fulvio Lizano Madriz for helping us through the project period.

Contents

1. Introduction

Overweight, obesity and other lifestyle diseases resulting from lack of exercise is a world-wide problem for humanity. Increased exercise could be a solution, but for this solution to succeed the exerciser has to maintain a good condition. Lack of condition, motivation and desire can be some barriers to bypass before exercise can be a part of your life. Your lack of motivation could be due to limited free time, stressful work or simply no will to exercise. An important factor for exercisers is motivation. Motivation can drive you to exercise on a daily basis and maintain a healthy body. Motivation, however, might not come easily to everyone. Many exercisers use music as a motivational element in their training and also to keep thoughts away from the body's exertion. Even though music can be a motivational factor it might not be utilized to its fullest. In addition, creating and managing exercise plans might be troublesome and if you are not aware of how an exercise plan should be structured this can be quite hard to do as well. Some users might also want to pick a selection of songs which gives them the most "drive" for each exercise plan. As of now users pick the music they want to listen to manually. Choosing the right music for each individual exercise session quickly becomes time consuming and might require you to do this additional times if you do not wish to listen to the same music every time.

This project researches how music can help exercisers doing exercise, inexperienced as well as experienced exercisers. A key element of this project is how music affects exercisers and how it may help to motivate or control the exercise. This project explains a system that can help a user to define an exercise plan and, from a selection of songs, select and play songs matching the user's exercise plan. The project concludes in a complete software prototype used to improve exercise sessions.

1.1. Problem statement

Exercisers, both professional and amateurs, who would like to schedule their exercises have to do so manually. This is done by keeping track of time with a watch or similar devices and thus knowing how far the exerciser is in the plan. Even though devices which signal a change in pace are becoming more frequent, this can be a troublesome process for longer exercise plans with many changes in pace. The problem can possibly be solved with the same solution for both amateurs and professionals, as the only difference in use is the number of repetitions and the pace for the different parts of the exercise plan. Another problem exercisers experience is lack of motivation while exercising, and some people use music to motivate themselves while exercising. Choosing music while exercising can interrupt your training and you might not want to listen to the same playlist over and over again. Making a new playlist for each exercise session and choosing the right music for each part of the exercise can be quite troublesome and time consuming.

We are a group of computer science students and the theme for this project is "user interaction". This means that the focus of this project is the direct interaction with potential users of the system designed in this project. We have formulated a problem statement that is as following.

How can an application assist a user in planning and executing an exercise plan using music as a means for the application to communicate with the user?

Part I.

Analysis

2. The task

In this first chapter the overall problem will be analyzed and the best way to solve it will be found. First the method will be chosen, then the effects of music on exercise will be described, a system definition will be found, already existing solutions will be looked at, and at last the context of the problem will be analyzed.

2.1. Purpose of the project

This project will try to help exercisers combine their own music with their custom exercise plans. An "Exercise-DJ" is a customizable music application that can improve exercisers interval training by playing music at an appropriate pace. For the system to choose the right song with the right rhythm it has to analyze the music in advance. The application's primary function will be to play the right music in relation to a customized training plan, though this function of the exerciser is independent of the manual time tracking. The ideal platform for this type of application would be a hand held device such as a smartphone or mp3-player, but in this project only a functional prototype for a PC environment will be developed.

2.2. The effect of music on exercise

The effect of music on exercise performance has been studied for the past 40 years, but many studies done before mid-1990 suffered from methodological shortcomings [13]. In 2012, Karageorghis and Priest [13, 14] collected and reviewed several different sources from 20 years of academic research about the effects music can have on exercise performance.

Karageorghis and Priest assert that most studies follow a similar conceptual model on how music can have *motivational qualities*. These four factors described are as follows [12]:

Rhythm response The effects on the musical rhythm, especially tempo.

Musicality The pitch-related elements of music such as harmony (how the notes are combined when played together) and melody (the tune).

Cultural impact The pervasiveness of the music within society or a sub-cultural group. The theme from *Rocky* is an example of a piece of music with powerful cultural impact.

Association The extra-musical associations that may be evoked. This can be on either a personal or cultural level. For example, the composition Chariots of Fire by Vangelis is often associated with Olympic glory.

Both rhythm and musicality are solely dependent on the music itself, and these are therefore classified as *internal factors*, whereas cultural impact and association are *external factors* as they are not necessarily dependent on the music itself [13].

These four factors rank from rhythm being the most important to association being the least important in determining the motivational qualities of a piece of music. These motivational qualities can then in turn influence the *arousal*; the amount of psychological excitement, the *rating of perceived exertion (RPE)*, and the mood during the exercise (see figure 2.1). Music can for example be used as a arousal control tool by either playing more intense music to increase arousal, or less intense music to decrease arousal. Reducing the RPE means that the person exercising feels less exhausted during, or after, training due to the distracting effect of music. In combination, these factors can be used to improve exercise adherence or as a pre-event routine.

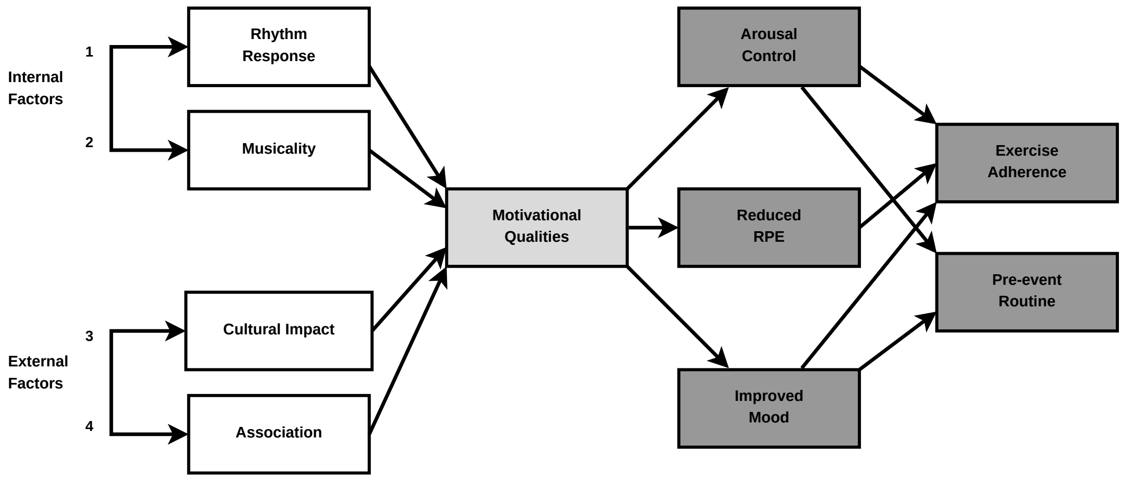


Figure 2.1.: Conceptual framework of motivational music by Karageorghis et al. [12]

In addition to these factors, whether motivational qualities actually lead to any major difference depends on the exercise. *Aerobic exercise*, light to medium exercise where the body uses oxygen to meet energy demands, saw the largest benefit of using music, while *anaerobic exercise*, intense exercise without the direct use of oxygen in the muscles, saw less benefit when using music as a stimulant [14].

In their review, Karageorghis and Priest [14] concluded from several studies that music can have a *ergogenic* effect, meaning that music can increase performance during exercise. Karageorghis and Priest also concluded that music is associated with a ~10% reduction in RPE (rating of perceived exhaustion). These effects are more pronounced in recreational exercisers than in professionals.

From their research, Karageorghis and Priest [14] concludes the following:

The evidence in this field demonstrates that music has the capacity to exert beneficial psychological and ergogenic effects across a wide range of exercise tasks and through a number of applications, pre-task, in-task and post-task.

2.3. Interviews

In order to gain an idea on how to utilize the benefits of music during exercise, we interviewed an expert and an experienced trainer in the subject. The interviews were semi-structured, meaning that they loosely followed a preset structure, but could diverge if necessary. These interviews form the basis of the requirements specification in section 2.4.

2.3.1. Marina Aagaard interview

Marina Aagaard, *Master of Fitness and Exercise* from University of Southern Denmark, is an external lecturer for Aalborg University. Furthermore, Marina Aagaard has written more than 20 books about health and exercise. All this makes her one of Denmark's leading experts on the subject [5].

As a starting point for the interview the expert was introduced to the problem described in section 1.1. She explained that music can be very motivational in itself and that combined with the right pace (appropriate to an exercisers running pace) the music will improve the exercise and probably reduce running time for the exerciser. A question about music with increased pace potentially damaging the body due to overexertion was asked, which to Aagaard answered that music with increased pace might increase the wear on the body a bit, however no scientific articles supporting this notion was known to her.

As a suggestion for the prototype, Marina Aagaard suggested to implement some preplanned exercise plans for exercisers without substantial knowledge of exercise planning. She also said that any preferences should be set before the training starts, as no one wants to fumble with any device while running.

In a follow-up interview conducted at a later time Marina Aagaard created a few interval training exercise plans to be used as default plans for the prototype. These exercise plans can be found in appendix A.

2.3.2. Susanne Stamhus interview

Susanne Stamhus is an experienced runner and trainer in the Aalborg-based running club AMOK. Stamhus acts as a trainer for new members of AMOK until they join a group within the club. Thus she has experience with both seasoned runners as well as beginners.

Stamhus uses a variety of running plans, but primarily interval training and Tabata training. Tabata is a special high-intensity interval training regimen where 20 second bursts of very intense exercise is followed by 10 seconds rest. She primarily runs marathons and longer distances called ultramarathons.

Stamhus usually uses music when running long distances and occasionally uses music on her shorter interval runs. She does not use music to achieve better performance, but just for entertainment. This is primarily because she feels too experienced to achieve any major benefit from music (which correlates with the findings described in section 2.2). However she still sees music as a motivational factor, especially for less experienced runners.

Stamhus uses a running watch to see when the interval changes. Stamhus noted that looking at the watch can be problematic, as it requires you to take the eyes off the road where you are running. This problem have led to a few accidents of running into the curbstone.

2.4. Requirements specification

As Aagaard pointed out, the developed system should appeal to inexperienced as well as experienced runners. It will do this by providing predefined settings for the inexperienced runner as well as the possibility for customization for the experienced runner. The system needs to provide the user with a way to easily change the preferences of an exercise plan to satisfy the needs of experienced runners. Manipulating exercise plans should also be intuitive so that inexperienced runners can experiment with planning exercises. Giving the user of the system advice on how to customize a plan to fit a specific level of intensity/experience would not be a requirement, as no general definition of intensity level exists, and it could differ from user to user.

As both Aagaard and Stamhus mentioned, visually interacting with the system during the exercise session should be kept to a minimum as to not annoy the user, and to reduce the possibility of accidents. If possible, all communication from the system to the user should be sound-based and not distract the runner visually in any way. This communication would consist of informing the user that the pace/intensity of the exercise session changed, that the session is over, and other relevant information that the exerciser would want to know during an exercise. If any communication from the runner to the system is needed, the ideal solution would be to accept voice commands as instructions. This, however, might not be ideal if the user is heavily exhausted. The communication could also be done with certain motion-based gestures, however it should be certain that the running motions of the user is not confused with these.

Communicating a change in pace/intensity could be done solely by changing music to a more intense/faster song. The system should therefore be able to accurately represent the desired pace throughout an exercise using the music that the user has provided for it. As no scientific evidence for physical wear as a result of highly increased running pace has been found, it should be left to the user to ensure that he/she is not pushing him/herself too hard, and safeguarding against this should not be a requirement to the system.

2.5. System definition

We established in section 2.2 that music can not only be entertaining, but have a positive physical effect during light to medium exercise. For many recreational runners and bicycle riders, exercising is synonymous with listening to music — mostly as entertainment, rather than for performance.

The opportunity here lies in using music not just as entertainment, but also as a motivation and feedback tool. Using the analysis from the previous chapters, we have designed two different ways of using music to assist during exercising.

2.5.1. System definition 1 (S1)

A system for exercisers that can generate and play a playlist with a specific tempo/intensity matching a predefined exercise schedule. This is done by analyzing a number of supplied music tracks and building a variety of metrics (for example such as tempo, energy, etc.) for each track. The user can then use the system to design an exercise plan and let the system generate a playlist that matches the plan. The system has to run on a mobile unit and has to be usable by recreational exercisers. S1 is illustrated in figure 2.2.

2.5.2. System definition 2 (S2)

A system that selects and plays music during the exercise based on the users performance metrics. A higher performance would result in more intense music and would in doing so signal to the user that he is performing better, and thus providing positive feedback. Like the first system definition, the system should be able to run on a mobile unit and should be usable by recreational exercisers. S2 is illustrated in figure 2.3.

2.5.3. FACTOR

These two system definitions are similar in their use of music, but different in the way it is used to motivate the exerciser. In order to gain a better overview of the two different definitions, it is necessary to use the FACTOR-model. FACTOR is a set of criteria described in the textbook for the course of system development [17]. The 6 criteria that FACTOR describes are as follows:

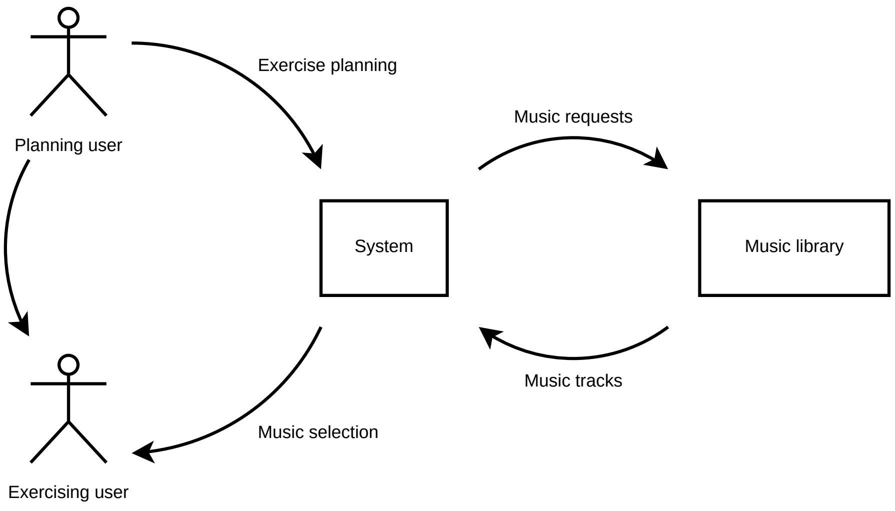


Figure 2.2.: Rich picture of S1

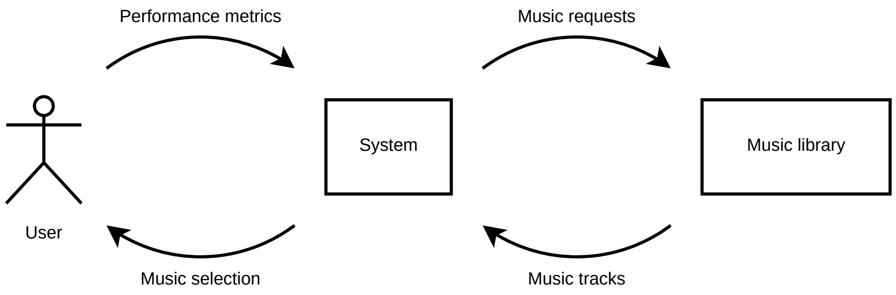


Figure 2.3.: Rich picture of S2.

Functionality The systems general functionality that support the application domain tasks

Application domain The parts of an organization that supervises, controls, or administrates a problem domain

Conditions The conditions for the development and use of the software

Technology The technology the system is developed on and the technology the system will run using

Objects The most important objects in the problem domain as seen by the user in the application domain

Responsibility The overall responsibility the system relate the its context

	S1	**S2**
F	Exercise planning and execution	Exercise feedback
A	Recreational exercisers (runners, bicycle riders, etc.)	
C	Exercisers with varying technological knowledge	
T	Smartphones, but also mobile electronic devices such as IPods	
O	Music tracks, playlists, exercise plan	Music tracks, performance metrics
R	Using music to plan exercises	Using music as a feedback tool

Table 2.1.: FACTOR analysis of S1 and S2

These criteria were applied to S1 and S2 as seen in table table 2.1. This uncovered several differences between the two system definitions. The most radical difference between these two proposed systems are the way their responsibility is shaped. In S1, the system is in control and the user follows the system whereas in S2 the user is in control and the system follows the user.

Both S1 and S2 meets the informants demands for minimum visually interacting with system (read all the requirements insection 2.4) by either changing music based on a predetermined plan (S1) or automatically changing music during to the feedback performance metrics (S2).

S2 has the drawback that it does not have a clear sense of control. The user influences the system with how well he/she performs, but the system also influences the user to some degree by playing faster/slower music as illustrated in figure 2.3. In principle, this can be seen as a feedback loop between the user and the system. This is in contrast to S1 where the user first relays information to the system, and then later receives information from the system during the execution of the exercise (see figure 2.2).

S1 is interesting from an interaction perspective because it utilizes music as a form of human–computer interaction. Sound is generally an underutilized sense in human–computer interaction and using music as a medium for communication is even less utilized. And S1 meets the other demand from the exercise expert, pre-planned exercise plans.

We have chosen to proceed with S1. This choice has primarily been made by the drawbacks as explained in the section above.

2.6. Existing solutions

A couple of existing systems similar to our system definition is available for the Android and iOS platform. These applications offers a variety of features such as music chosen depending on your pace and exercise planning tools. This section will describe some of the competing software solutions and their limitations.

2.6.1. PaceDJ

PaceDJ is a piece of software which allows you to input your desired pace in BPM (Beats per Minute) or measure this by using the smartphone's accelerometer while exercising. From the BPM inputted in PaceDJ, the application will find songs from your media library matching the desired BPM. The application can be configured to allow the use of songs with a BPM half or double that of the desired as they are in harmony with the tempo.

PaceDJ can determine the BPM of your songs using data from The Echo Nest online service which allows correct and fast analysis of songs, the application additionally offers a "virtual drum" to allow the user to correct mismatches in BPM recognition by manually tapping the beat of the song.

PaceDJ has a feature called BPM Shifting, which can increase or decrease the pace (BPM) of you own songs slightly to match the desired pace[19].

Limitations of PaceDJ

PaceDJ has some limitations in the way it is designed. It does not allow the user to create an exercise plan to follow. It does only allow for the selection of a specific BPM or to follow the exercisers current pace. Also it requires access to the Internet in order to analyze the tracks. Some features are only available for the iOS platform.

2.6.2. DjRun

DjRun offers all of the features already mentioned in PaceDJ. Furthermore DjRun includes a feature allowing one to configure or use pre-made training programs, allowing for planed exercises such as interval training. The plan can be made with both immediate pace changes, but also with gradually changing pace (see figure 2.4). This is possible as the application offers a dynamic regulation of BPM by speeding up or slowing down a song, this is done without pitch shifts to avoid voices being distorted. DjRun can also analyze songs offline using an analyzing algorithm or by making the user tap the beat manually.

Limitations of DjRun

DjRun has some issues with the pro version of the app not registering properly for some users. The app also does not find music that is stored on the phone's SD card.

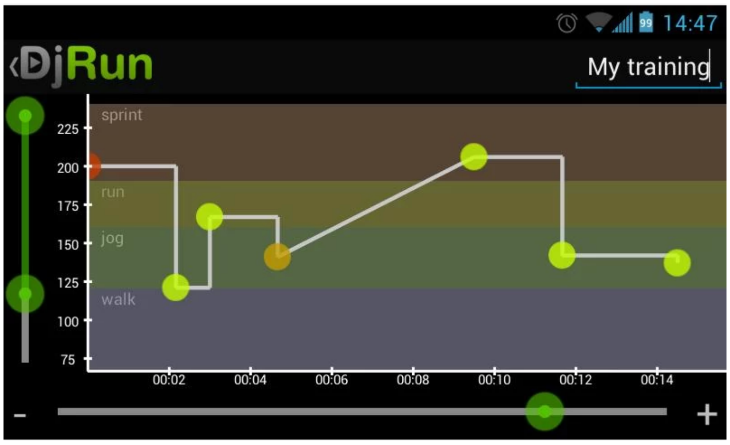


Figure 2.4.: DjRun exercise planner

2.6.3. Jog2Beat

Jog2beat is a freeware application with basic features like that of PaceDJ and DjRun; it can play music matching your workout speed. Jog2Beat does not analyze your music locally but matches the songs using an online service. Compared with the other alternatives, described above, Jog2Beat is a simpler application with less functionality. As the screenshots show (see figure 2.5), there are some predefined workout plans and it is not possible to edit or create new workout schedules, only specify a starting pace and the exercise time.

Limitations of Jog2Beat

The Jog2Beat solution is very basic and does not support the making of custom exercise plans or dynamic BPM regulation. Also the application needs an active Internet connection.

2.6.4. Conclusion

Most of the existing applications we looked at did not support the making of a custom exercise plan. Also, most of the systems require access to the Internet to be able to analyze songs from your music library. This means that the software is required to cache data for songs from the Internet in order to use them offline.

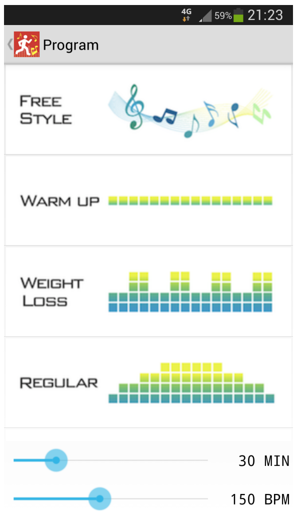


Figure 2.5.: Screenshot from the Jog2Beat application

2.7. Context

The application should be used on a mobile device while the user is exercising. It should allow the user to create an exercise plan, and it should play music according to this plan. By being available on a mobile unit, the user can exercise with the software right at hand.

2.7.1. Problem domain

The system should gather information about the tempo and intensity of a music track. The information gathering can be done in several ways. One way is to analyze the track's data directly by using an algorithm to find the tempo and intensity of the track. Another way to get the information is to collect it from an existing source, e.g. an online service like Echo Nest described in subsection 2.6.1.

2.7.2. Application domain

The system should help motivate exercisers and help them keep track of their exercise plans. The system should be available on a modern mobile unit from an application provider (i.e. Appstore/Play Store). The user creates and configures a desired exercise plan and the system saves it for later use. The user can also select from any of the saved exercise plans to use and track the progress of the plan. The system should play music matching the intensity of the current point of the exercise plan. The user can use specific buttons to skip the current song and play the next song matching the current intensity.

3. Problem domain

In this chapter the problem domain for the system is analyzed. First the overall structure will be described, and then the classes will be analyzed. Lastly the events in the system will be described.

3.1. Structure

The diagram shown in figure 3.1 is the class diagram of the system. This diagram demonstrates the relation between the different parts of the problem domain and how they relate to each other. In addition, it shows the multiplicity, which is a measure of how many classes are related to a given class.

The ExercisePlan class is responsible for saving information about the users different exercise plans. The TrackList is responsible for storing information about a collection of music tracks. The Track class is responsible for storing information about a single music track. The ExerciseSession class is used to store information about an exercise session consisting of exactly one TrackList and one ExercisePlan. A TrackList or an Exercise-Plan can be used by several different ExerciseSessions, hence the 0..* to 1 multiplicity between TrackList and ExerciseSession and between ExercisePlan and ExerciseSession. A TrackList can store information about none to many songs and a Track can be a part of several different TrackLists, which leads to a 0..* to 0..* multiplicity between Track and TrackList.

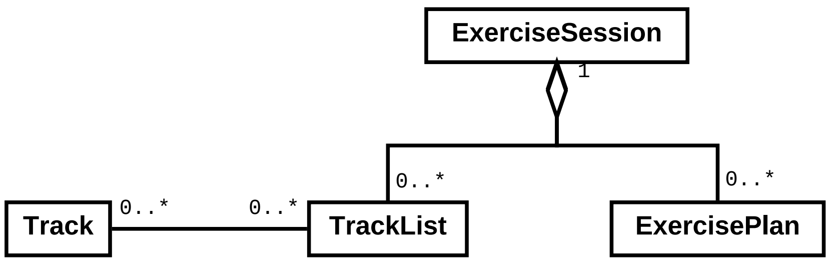


Figure 3.1.: Class diagram

3.2. Classes

As demonstrated in section 3.1, the system consists of four classes; Track, TrackList, ExercisePlan, and ExerciseSession.

3.2.1. Track

Definition

This class is used to store information about music tracks. This information can be metadata like the artist, tempo, energy, duration, etc.

Behavior patterns

In figure 3.2 is the state diagram for the class Track shown. A state diagram shows single objects various states and how actions change the state that they are in. When a user adds a music track to the system a Track object is created. The Track can then either be removed or analyzed. If the Track is analyzed it becomes active and can be used during an exercise session. From the Active state the Track can either be removed or added/removed from a TrackList. Removing the Track itself removes it entirely from the system.

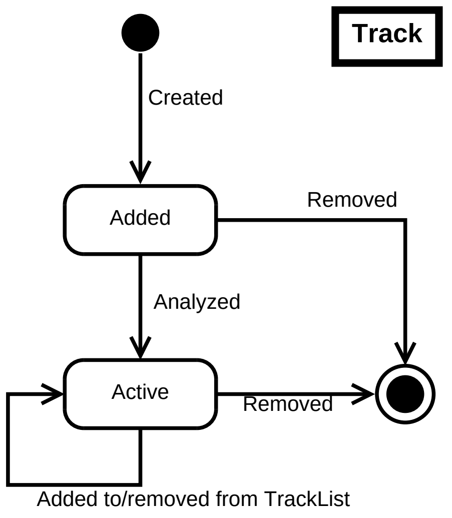


Figure 3.2.: The state diagram for the Track class

3.2.2. TrackList

Definition

The TrackList is a collection of tracks and is used to define what music tracks can be played during an exercise session.

Behavior patterns

figure 3.3 shows the state diagram for the TrackList class. When a user creates an empty collection of tracks it is stored in a new TrackList object. The TrackList starts in its only state; Active. From here the TrackList can have music tracks added to it or removed from it. The TrackList itself can also be removed leading to its removal from the system.

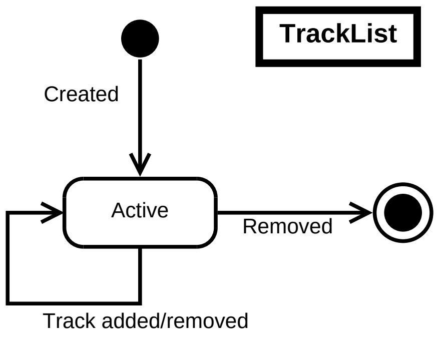


Figure 3.3.: The state diagram for the TrackList class

3.2.3. ExercisePlan

Definition

This class stores information about an exercise plan; the duration of the warm-up and cool-down, the duration of high and low-intensity intervals as well as the number of repetitions in the interval training.

Behavior patterns

figure 3.4 shows the state diagram for the ExercisePlan class. When a user creates a new exercise plan, the plan will be represented in the system as an object of the type ExercisePlan. In its creation the ExercisePlan immediately becomes active and is in a Stopped state. From this state the object can either be started or removed. Starting the ExercisePlan sends it to the Playing state. From here the only option is to stop the ExercisePlan either by user input or by end-of-plan event. Removing the ExercisePlan object causes its termination.

3.2.4. ExerciseSession

Definition

This class stores information about the progress of an exercise session and consists of a TrackList and an ExercisePlan.

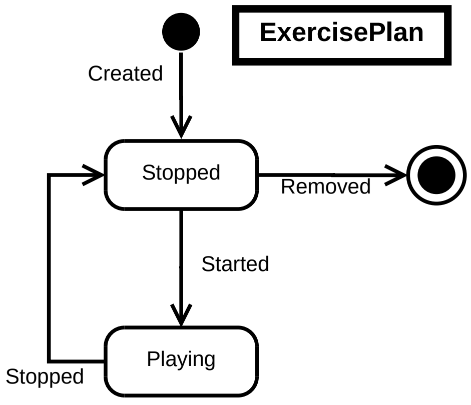


Figure 3.4.: The state diagram for the ExercisePlan class

Behavior patterns

figure 3.5 shows the state diagram for the ExerciseSession class. When a user starts an exercise session an ExerciseSession class is created and put in the Stopped state. The ExerciseSession can then be started/resumed or ended. If the ExerciseSession is started or resumed it changes to the Started state. From here it can either be stopped/paused or ended. Stopping/pausing the ExerciseSession from this state returns it to the Stopped state. Ending the exercise session from either the Stopped state or the Started state removes the ExerciseSession object from the system. The exercise session is ended by completing the associated ExercisePlan or by the user actively ending the session.

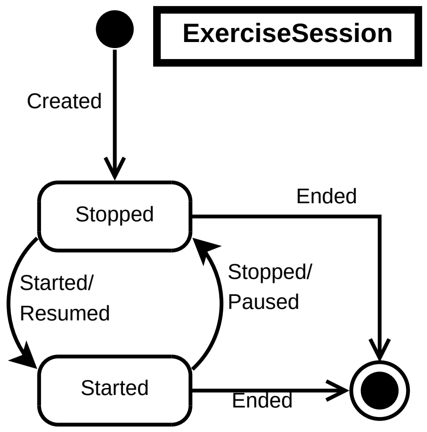


Figure 3.5.: The state diagram for the ExerciseSession class

3.3. Events

Table 3.1 shows the list of events in the system.

	Classes			
Events	Track	TrackList	ExercisePlan	ExerciseSession
Track added	x			
Track removed	x			
Track added to/removed from TrackList	x	x		
TrackList created		x		
TrackList removed		x		
ExercisePlan created			x	
ExercisePlan removed			x	
ExerciseSession created		x	x	x
ExerciseSession started				x
ExerciseSession stopped				x
ExerciseSession ended				x

Table 3.1.: Event table

4. Application domain

This chapter will analyze the application domain. It will look at the usage and the various actors. It will look at specific use cases and will describe the various functions of the system. It will also describe the user interface and the technical platform.

4.1. Usage

4.1.1. Overview

Table 4.1 shows the actor table for the system as well as what actor interacts with the use cases. The planner and the exerciser actors both have one main use case of the system.

	Actors	
Use cases	Planner	Exerciser
Manage exercise plan	x	
Start exercise		x

Table 4.1.: Actor table showing use case interaction

4.1.2. Actors

In order to better visualize how users interact with the system, we have created a set of actor specifications. An actor can be seen as a user's interaction with a specific part of the system. Thus, a single user can be multiple actors depending on the user. The system contains two actors; a planner and an exerciser. The planner plans and creates exercise plans and the exerciser start exercise plans. This could be a single person planning and using his own exercise plans or two different persons, for example an experienced exerciser planning exercise plans for a non-experienced exerciser.

4.1.2.1. Planner

Purpose

The purpose of the planning actor is to create exercise plans to suit an exerciser's level of exercise. The planner needs an easy to use user interface to use the planning functions of the system.

Characteristics

The planner is a person with highly varying experience with planning efficient exercise plans

Examples

Planner A is a student who goes for an early morning run, but lately has not been working out at an intensity that suits her needs. She cannot run without listening to music and therefore needs a tool to plan her exercises more to suit her running needs. She does not have any experience with planning a complex exercise session.

Planner B is a running instructor in a gym. Every week he helps several clients as well as instructing a couple of running programs. He needs an efficient planning tool with features and professional terms he can relate to. He has a lot of experience with how an exercise session should be pieced together and does not want the tool to interrupt him when he is using it.

4.1.2.2. Exerciser

Purpose

The purpose of the exercising actor is to exercise following a specific exercise plan while listening to music fitting the intensity of the exercise session.

Characteristics

The exerciser does not necessarily have any knowledge about music intensity levels or tempo, and therefore does not know when to play certain music tracks to get the desired level of motivation to accompany the exercise session.

Examples

Runner A is a daily runner with a broad taste of music. He feels more motivated to run when he listens to the right kind of music and needs a tool to play this kind of music in accordance to his preferred exercise rhythm.

Runner B is an athlete runner. To improve her skills she must run a lot of different interval training sessions a day. To help with motivation she listens to music while running but she must also keep an eye on a stopwatch that tells her when to start sprinting and when to slow down. The music also does not fit when she speeds up and is therefore degrading her performance. She needs something to tell her when to run at a high pace and when not to while playing music that fits the desired tempo of the intervals.

4.1.3. Use cases

figure 4.1 shows the interaction between the actors and the use cases of the system. Two use cases concerns the exercises of the system; “Manage exercise plan” and “Start exercise”.

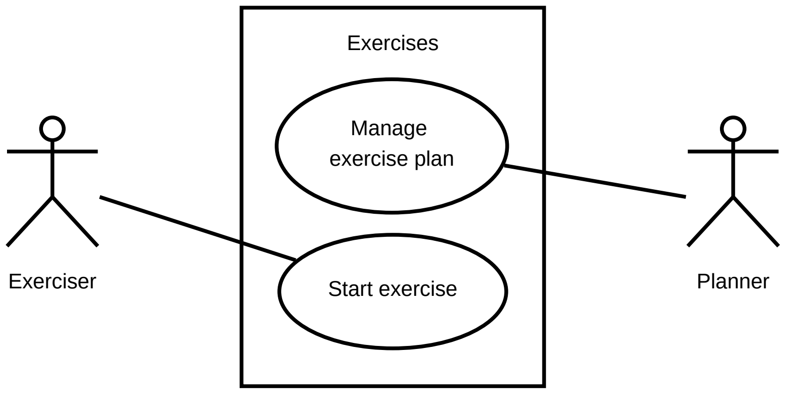


Figure 4.1.: The use case diagram of the system

Manage exercise plan

Managing exercise plans consists of adding, removing, and editing. When a user wants to create a new exercise plan he/she would first customize the intervals of the plan, choosing the wanted options for warm-up and cool-down duration, the number of high-intensity intervals to run in between the optional warm-up and cool-down as well as the duration of the high and low-intensity intervals. Then the user names the exercise plan and saves it. When a user wants to edit an existing plan, he/she would edit the values of the plan and save the changes. Removing a plan simply requires the user to confirm the action resulting in the removal of the selected plan from the system.

Start exercise

When a user wants to start an exercise session he/she would first choose from a list of available exercise plans and also choose a playlist to determine which songs should be used with the selected exercise plan during the session. Now the user can start the exercise with the assigned button and the system will handle the playback of music. If the user wants to he/she can skip the current playing track at any time and the system will find another suitable song. The exercise stops when the plan reaches the end or when the user signals the system to stop the session.

4.2. Functions

The system is responsible for creating an exercise plan and finding music to fit. The systems functionality will therefore reflect this purpose. Primarily, the system should include functions that allow it to manage different music tracks in a library, to analyze these tracks and find an intensity level for them, to create an exercise plan, and to match this plan with the previously analyzed music. This section will show a list of the desired functionality in the system as well as the complexity of these functions. It will also provide further detail on the most complex functions.

4.2.1. Complete list of functions

Table 4.2 shows the complete list of functions for the system.

Function	**Complexity**	**Type**
Analyze music track	Medium	Calculation
Load tracklist	Simple	Update
Find music track from tracklist	Simple	Read
Get metadata from music track	Simple	Read
Create exercise plan	Simple	Update
Start exercise plan	Simple	Read
Find best fitting music track	Semi-Complex	Calculation
Skip song	Simple	Update

Table 4.2.: List of functions

The system has many simple functions. Most of them just updates or reads from the model, and requires no calculations of any kind. These include loading a track list, retrieving a track from the track list, getting metadata from a music track, creating an exercise plan, starting an exercise plan, and skipping a song. There are two functions that are a bit more complex and require a calculation. The first is to analyze a music track. The analysis of a music track should be a calculation made based on the tempo of the music track (beats per minute) and possibly the intensity level of the music track. Both values should be read from the metadata of the track, and put through the calculation to categorize the track. The second is to find the best fitting music track. The best fitting music track is the track which tempo/intensity rating best matches the desired tempo/intensity rating provided to the function. The track should be found on a certain track list and could additionally omit previously skipped tracks.

4.3. User interface

The user interface should incorporate the ability to follow the user through the process of creating, editing, and removing an exercise plan as well as being visually assisting when the user wants to start and run an exercise session. The interface to the user should also be touch-friendly, as it is to be used on mobile devices where touch-screen is the main input method. That leads to requirements such as big buttons and easy-to-follow menu flow and non-ambiguous icons.

4.3.1. Form of dialogue

As the user interface should be used with mobile devices with relatively small screens, a window based dialogue is out of the question. What Gong and Tarasewich [8] proposes, is to use multilevel and hierarchical mechanisms. This could be to a simple system of menu-stacks that all root in a main menu screen. This main menu should have four options; start exercise, create exercise, options, and exit application. All of these four options should be depicted as button controls as they all represent major operations leading to completely new screens.

The create exercise sub-menu stack should guide the user through a possible series of screens containing input-controls that needs to be adjusted to create an exercise plan. The navigation through these optional screens should be obvious to the user with forwards and backwards buttons at the bottom of the screens. The phone's physical back button should prompt the user to return to the main menu without saving. The screens could contain a progress bar showing how far the user has come in the creation process, as guided by Gong and Tarasewich [8].

The start exercise sub-menu stack should present the user with a list of available exercise plans to choose from. The list should only consist of the title of the available plans for simplicity. This screen should allow the user to select the track list to use during the session as well as allow the user to edit or remove the available plans. After selecting both a plan and a track list the user could proceed to load the plan. When the plan is loaded the user is presented with the play screen. From here the user can see his/her progression in the plan, information about the song currently being played, if any, skip the current song, and start/pause/stop the exercise.

The options sub-menu screen should contain a selection of sub-menus and controls used to tweak the functionality of the system to the users preferences. The controls could consist of check-boxes, combo-boxes and sliders.

Table 4.3 shows the complete list of components that the user interface will use. These components have been chosen to best fit the small screens of mobile devices and the touch input. Activating a text-box would pop-up the mobile devices on-screen keyboard for inputting text. This keyboard would take up a lot of screen space and text-boxes should therefore, if possible, be given their own dedicated screen or context. The components have been chosen based on the argumentation presented in this section.

Furthermore as Gong and Tarasewich [8] points out, a mobile phone is a very personal device, and the applications should therefore also allow personalization to some extent.

Menus	Buttons	Tiles
Combo-boxes	List-boxes	Scroll-bars
Check-boxes	Text-boxes	Sliders
Indicator-diodes	Pop-up menus	Labels

Table 4.3.: Complete list of components in the user interface

4.3.2. Overview

The navigational flow of the user interface can be seen in figure 4.2. The GUI is started in the main menu where the user has four simple options; start, create, options, and exit. The first three options leads to the corresponding sub-menu screens and each of these screens leads back to the main menu through progression or cancellation.

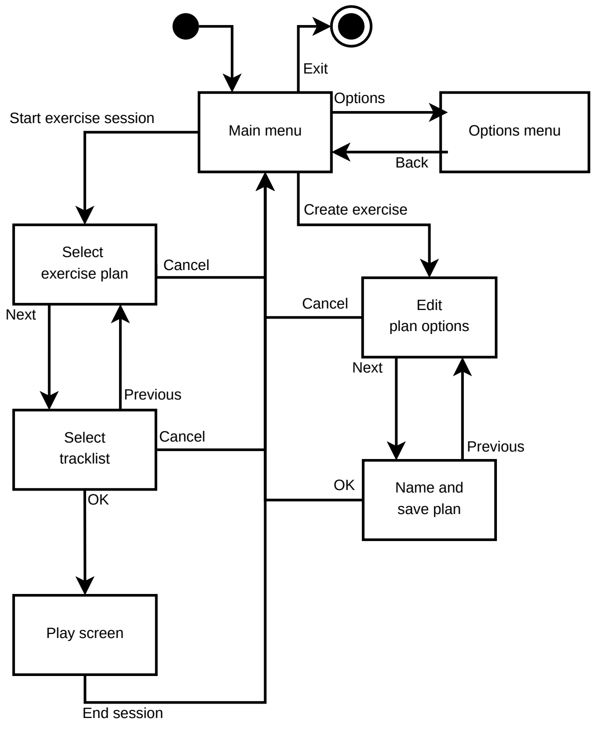


Figure 4.2.: The navigation diagram of the user interface

4.3.3. Examples

This section will cover conceptual design examples of the graphical user interface. Here two examples of concept designs are described and visualized; the process of creating a new exercise plan and the process of starting an exercise session.

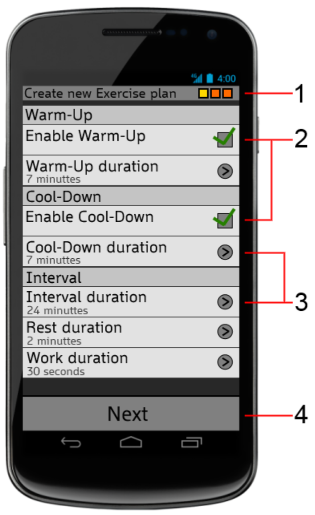
Create new exercise plan

The first screen the user meets when choosing to create a new exercise plan is seen in figure 4.3a. Label 1 shows where the title-bar of the screen is placed as well as an indicator that shows that this is step one of all the steps. Label 2 shows examples of check-boxes and label 3 shows examples of multi-choice controls. When pressing a tile with the multi-choice icon, the little arrow in a circle, a pop-up menu appears in the middle of the screen taking focus and listing the available options for that control. Here the options could consist of a list of predefined time intervals or a slider that the user can manipulate. Label 4 shows the button control to proceed to the next step and the next screen. As this is the first of three screens there is not a button for the previous screen. The user can instead use the phones "back" button located below the screen to return to the main menu after a prompt telling the user that his/her changes will be lost.

Start exercise session When the user chooses to start an exercise session he/she is greeted with the screen seen in figure 4.3b. Label 1 outlines a exercise-tile element. The tile represents an exercise plan and displays relevant information about the plan. Label 2 shows the name of the exercise plan. Label 3 shows the name of the default playlist to use with the exercise plan. If no playlist has been chosen as the exercise plan's default playlist, nothing is displayed informing the user that a playlist must be chosen before the plan can be started. Label 4 shows the total length of the exercise plan. The phones "back" button returns the user to the main menu.

4.4. Technical platform

Due to project requirements, the system is initially developed for a PC environment and programmed in C#. A final system, however, would have to be ported to mobile units as well. The user interface will be based on menus and windows, and will be operated with a mouse and keyboard. Touch controls will not be implemented in this prototype.


Create new Exercise plan

Warm-Up

Enable Warm-Up ✓

Warm-Up duration
7 minuttes

Cool-Down

Enable Cool-Down ✓

Cool-Down duration
7 minuttes

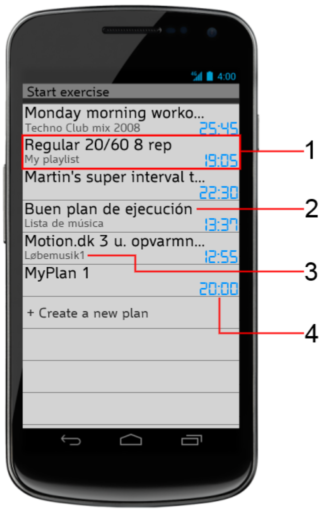
Interval

Interval duration
24 minuttes

Rest duration
2 minuttes

Work duration
30 seconds

(a) The first screen of the “Create new exercise plan” process


Start exercise

Monday morning worko... Techno Club mix 2008	25:45
Regular 20/60 8 rep My playlist	19:05
Martin's super interval t...	22:30
Buen plan de ejecución Lista de música	13:37
Motion.dk 3 u. opvarmn... Løbemusik1	12:55
MyPlan 1	20:00

+ Create a new plan

(b) The “Start exercise” screen

Figure 4.3.: Examples of Graphical User Interfaces

5. Recommendations

There are several factors to look into when deciding how to implement a system. This chapter will look into the realizability and strategy of the system implementation.

5.1. The realizability of the IT-system

The central point of the system is to be able to select music using certain metrics (such as tempo) for the users exercise. Acquiring the tempo, called *beat tracking*, from a piece of music is non-trivial and there are several different methods to do so [9]. Implementing an algorithm for this is possible, but would require a significant amount of time to implement and ensure that it provides good results in most cases can be difficult.

Another option is using a service-based model where a central service analyzes and caches information about a music track. Such services already exist due to the rise in music streaming services over the last decade. *The Echo Nest* is an example of such a service [2]. Services such as these require less computation from the mobile device, but instead require an Internet connection instead.

Both of these methods have been used by similar systems in the past (see section 2.6), so both are viable options. We currently do not have any experience with the signal-processing necessary for beat tracking, so our initial implementation will use the service-based model, though we should strive to make this implementation polymorphic so it is possible to implement other beat tracking methods if needed.

We have in earlier projects gained experience with .Net application development along with smaller database systems, so we assess that we have the required technical experience to build such a system.

5.2. Strategy

Our work will primarily be towards a Windows-based .Net prototype. We have decided to use Windows Presentation Foundation (WPF) as our user interface implementation, as it is a much more powerful GUI platform than Windows alternatives such as Windows Forms [4] and provides build-in methods of playing music eliminating the need to implement or use another library supporting playback of music files.

Part II.

Design

6. The task

This chapter describes the purpose of the design document. It also describes how some of the problems and aspects of the analysis document is not taken into the design phase and what will be done instead. This chapter also defines how the priorities of the different design criteria have been set in order to fulfill the requirements specified in the analysis.

6.1. Purpose

The designed system should help increase motivation and enjoyment of exercise. The system should help exercisers to plan and execute exercises aimed at interval training. It should also be able to play music contained in a playlist during the exercise in a way that enriches the experience for the user. The user should be able to tweak the systems method of choosing what an enriched experience is and should also be able to influence the choice of music in some way during the exercise session. The system should therefore not have a plan set in stone before an exercise session is started, but instead only load the necessary resources to begin a session and then systematically progress through the plan only if the user does not intervene.

6.2. Aversion from the analysis

As the prototype is developed for a PC environment, several aspects of the analysis is not taken into consideration when designing the program. The analysis can be seen in details in chapter chapter 3 and chapter 4.

The analysis mentions a Track and TrackList class for storing information about music tracks and collections of music tracks respectively. In this prototype a Track will be represented as a string of text pointing to the location of the music file on the local machine running the program. No information about the music track, other than the file's own metadata, will be stored by the program when managing tracks. A TrackList will, in the prototype, also just consist of a list, or similar collection type, of these text strings that points to individual music files. Analysis of the individual music tracks will happen on run-time, when the program starts, by the functionality-layer and the analysis-data will be stored in temporary memory when the user requests to use a specific track list. This is instead of analyzing the individual files once when they are added to the program and save the analysis on disc. A track list will be creatable from existing playlists of the .m3u format, but will not be manageable by the prototype.

The ExerciseSession described in the analysis keeps track of the progress of an exercise plan and contains a reference to a track list containing the music tracks to use during the exercise. This functionality will be shared between the functionality-layer and the interface-layer of the program. The user interface will keep track of the progress through an exercise plan as well as playing the songs and the functionality-layer will be responsible for finding out which songs to play next.

6.3. Quality goals

Table 6.1 shows the priority of the general design criteria in the development of the system. These criteria have been set to help direct the development process prior to the actual programming of the system and are therefore a projection of the system's final strengths and weaknesses. The importance of each criterion is a goal/limit for how much attention the criteria should receive during the development of the system. The priorities have been set like seen in table 6.1 on the basis of the analysis and the task at hand to reflect the importance of each criteria in the context of the system.

Criteria	**Very important**	**Important**	**Less important**	**Irrelevant**	**Easily fulfilled**
Useable	✓				
Secure					✓
Efficient		✓			
Correct		✓			
Reliable			✓		
Maintainable			✓		
Testable			✓		
Flexible		✓			
Comprehensible		✓			
Reusable				✓	
Portable		✓			
Interoperable				✓	

Table 6.1.: Priority of the design criteria

The main area of focus in the system is that it must be usable. The system should meet the requirements defined in the analysis and the expectations of the potential users. Another important criteria to the system is flexibility; as the system would be used by a larger and broader set of users than what the system-requirements in the analysis where based on, it should maintain high flexibility to make sure that changes to the system would not be costly. The comprehensibility of the system is also of a certain importance as several people will work on the development of the system simultaneously and all need to easily understand what the individual components are responsible for. As the system is developed for a desktop environment but is a prototype of a mobile application it is also important that the system is portable to some extent. The security of the system, albeit not irrelevant, is so easily fulfilled that it should not be given any resources. There simply is not anything of any importance in the system to secure. As the system will be a stand-alone prototype defining a possible end product, the reusability and interoperability of the system is deemed irrelevant.

7. Technical platform

In this chapter will the technical platform for the prototype be described. This includes equipment, programming software, system interfaces and design languages.

7.1. Equipment

The system is designed for mobile devices, such as smartphones, iPods etc., so the exercisers can use the system while they are running or doing other exercises. This necessitates some considerations on the efficiency of the application, as mobile devices are not as powerful as desktop computers. The prototype will, as previously explained, be developed for a PC environment.

7.2. Software

As part of the requirements of this project, it is a requirement to write the program in C#. C# is an object-oriented programming language with a wide array of libraries from both Microsoft and other contributors.

For developing the GUI, this project's prototype will be using the Windows Presentation Foundation (WPF), a Microsoft-developed GUI system based on the .Net Framework.

This projects prototype will be coded with Microsoft Visual Studio, mainly because of its C# and WPF support.

In order to identify songs, we either need to analyze them or use the metadata stored in the music files. We chose the latter, as it is significantly simpler to do so. To read these metadata tags we use TagLib, an open-source .Net library for reading and editing metadata for several popular audio and video formats.

For serialization to and from the hard drive, we use the Entity Framework from Microsoft. Entity Framework is a system that allows interfacing with databases using C#. This means that writing SQL is unnecessary, as the Entity Framework can generate valid SQL code from C# code. This abstracts away the database and allows us to use the exact same code for different databases. We have chosen to use SQL Server Compact Edition by Microsoft. This is primarily because it is easy to integrate with Visual Studio.

7.3. System interfaces

Beside the mobile device, the system needs an operating system on which to run. This operating system should, in the final version, be compatible with some mobile devices and could be Apple's iOS, Google's Android, or Microsoft's Windows Phone. But the prototype developed in this project will only be runnable on Windows-platforms because it is currently the only platform that supports the WPF framework.

In addition, the application uses SQL Server Compact Edition (SQLCE) which is only available on Windows. However, any database solution supported by the Entity Framework can be used instead, and it would therefore be easy to implement other databases if this is needed for a specific platform.

The application needs to gather information about music in order to choose what music tracks to play during an exercise. To do this, it uses The Echo Nest — a large web service that contains information about hundreds of thousands different pieces of music (see section 9.1 for more information about the use of The Echo Nest).

7.4. Design language

This project's design is described with the Unified Modeling Language (UML), as it provides a large standardized selection of object-oriented constructs that can be used to plan and design object-oriented software.

8. Architecture

A system must have a proper architecture design before it can be implemented. This chapter will look at the architectures of the systems components and processes.

8.1. Component architecture

The component architecture of the system is based on a layered architecture. As we are working with C# in Visual Studio, we have chosen to define a component as a single *project* within the Visual Studio solution for our prototype. In our prototype, components can only have dependencies further down in the layer hierarchy though it is not necessarily to the components directly below. Mathiassen et al. [17] defines such a system as *open-strict*. This particular implementation of a layered structure closely resembles the *Generic Architecture Pattern* defined by Mathiassen et al. [17], with the addition of a "third party service" layer used for communicating with The Echo Nest.

The architecture of the system is seen in figure 8.1. At the top layer is the graphical user interface (GUI) component which depends on the majority of the remaining components in the system. Below the GUI is functionality, which includes `Prototype.Media,` that handles selection and grouping of music tracks, and `Prototype.KMeans` which is responsible for implementing the k-means clustering algorithm described in chapter 11 and chapter 19. At the model layer is `Prototype.Exercise` which is responsible for modeling exercise plans. In addition, a cache for The Echo Nest is implemented in `Echonest.Cache` in the model layer (see section 9.1). At the technical platform level, we use Windows Presentation Foundation (WPF), TagLib, and the entity framework (`System.Data.Entity`). In addition, every component implicitly depends on the .Net Framework (not displayed on figure 8.1 for brevity). The single third party service is `Echonest` which communicates with The Echo Nest through an Internet-based API.

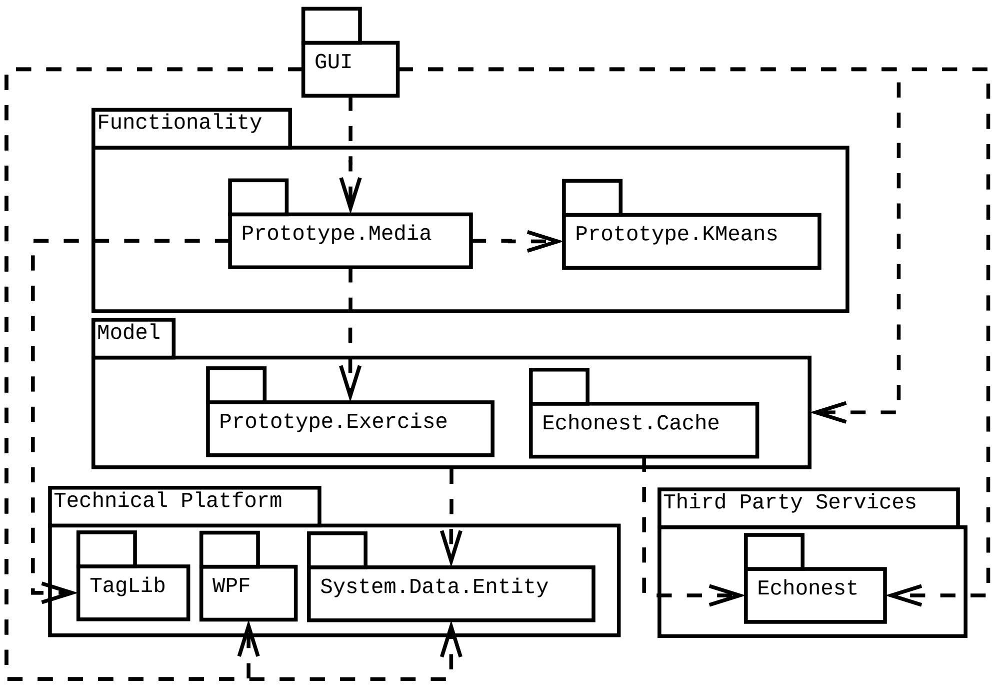


Figure 8.1.: The system's component architecture

8.2. Process architecture

The system should only be in use by one user at a time. Playing back music, even fading between several music tracks, should not require separate threads to maintain and could even be handled by the user interface. Finding a new or the next song to play, should also not require usage of threading if the information about all possible music tracks has been loaded into memory prior to starting an exercise session. To do so, all possible tracks should be categorized by contacting The Echo Nest cache. This could take some time if the cache needs to contact The Echo Nest service, but would not necessarily need to be executed asynchronously or in another thread. This possible waiting time could be hidden behind a loading screen when the user chooses to start an exercise session.

8.3. Standards

Even though the developed prototype will be designed primarily for a Windows platform, the application as a whole is intended for use on mobile devices. The prototype should therefore try to incorporate at least some design guidelines and standards for mobile applications. The design of the user interface will follow the most applicable official Android design principles, styles, and patterns as described in the Android design standards [1]. This includes a recommended height for all buttons and input-controls of 48dp, using standard icons for different buttons, keeping to one theme throughout the program, and keeping an easy-to-follow navigation between screens.

8.4. Class diagram

To visually describe the object-oriented system designed a UML class diagram has been made. This class diagram is seen in figure 8.2 and shows the different classes in the system and what components they belong to. It also shows all the relations between the classes of the system and follows the architecture designed in section 8.1. chapter 9 describes the functionality of the different components as well as some of the important classes.

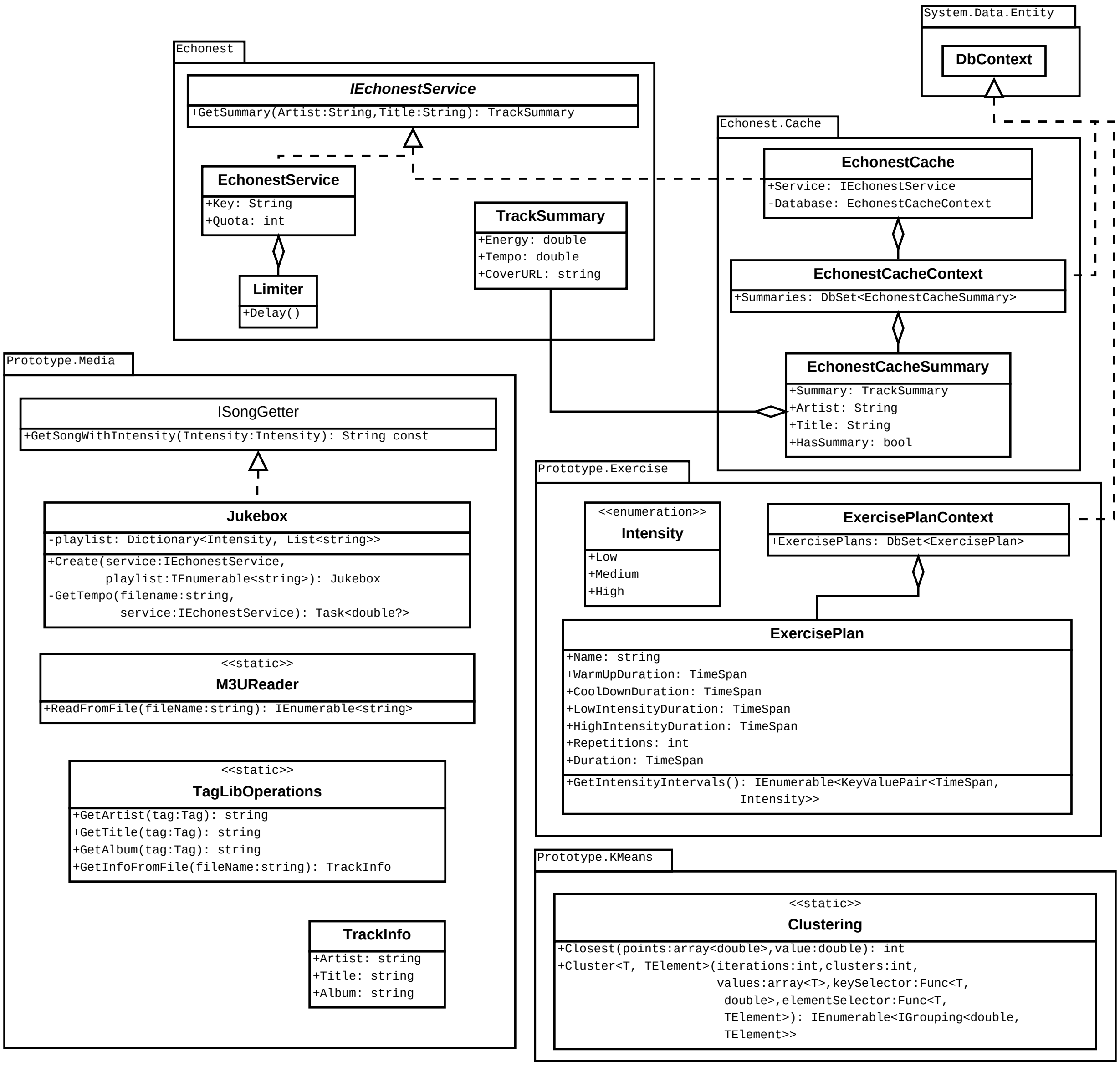

System.Data.Entity
DbContext
Echonest
IEchonestService
+GetSummary(Artist:String,Title:String): TrackSummary
EchonestService
+Key: String
+Quota: int
Limiter
+Delay()
TrackSummary
+Energy: double
+Tempo: double
+CoverURL: string
Echonest.Cache
EchonestCache
+Service: IEchonestService
-Database: EchonestCacheContext
EchonestCacheContext
+Summaries: DbSet<EchonestCacheSummary>
EchonestCacheSummary
+Summary: TrackSummary
+Artist: String
+Title: String
+HasSummary: bool
Prototype.Media
ISongGetter
+GetSongWithIntensity(Intensity:Intensity): String const
Jukebox
-playlist: Dictionary<Intensity, List<string>>
+Create(service:IEchonestService, playlist:IEnumerable<string>): Jukebox
-GetTempo(filename:string, service:IEchonestService): Task<double?>
<<static>>
M3UReader
+ReadFromFile(fileName:string): IEnumerable<string>
<<static>>
TagLibOperations
+GetArtist(tag:Tag): string
+GetTitle(tag:Tag): string
+GetAlbum(tag:Tag): string
+GetInfoFromFile(fileName:string): TrackInfo
TrackInfo
+Artist: string
+Title: string
+Album: string
Prototype.Exercise
<<enumeration>>
Intensity
+Low
+Medium
+High
ExercisePlanContext
+ExercisePlans: DbSet<ExercisePlan>
ExercisePlan
+Name: string
+WarmUpDuration: TimeSpan
+CoolDownDuration: TimeSpan
+LowIntensityDuration: TimeSpan
+HighIntensityDuration: TimeSpan
+Repetitions: int
+Duration: TimeSpan
+GetIntensityIntervals(): IEnumerable<KeyValuePair<TimeSpan, Intensity>>
Prototype.KMeans
<<static>>
Clustering
+Closest(points:array<double>,value:double): int
+Cluster<T, TElement>(iterations:int,clusters:int, values:array<T>,keySelector:Func<T, double>,elementSelector:Func<T, TElement>): IEnumerable<IGrouping<double, TElement>>

Figure 8.2.: Class diagram

9. Implementation

This chapter will document some of the development of the prototype. Relevant code examples as well as figures and models will be used to describe the prototype implementation. The complete source code can be found on the attached CD-ROM.

9.1. The Echo Nest

The Echo Nest is a company that provides intelligent solutions for analyzing and recommending music and exposes it through a web-based API. The Echo Nest automatically updates their music database which contains data for more than 34 million songs from more than 2.4 million artists. A number of large media corporations, such as MTV and Spotify, use The Echo Nest service for music recommendation, playlist generation, and music identification[2].

The API used to communicate with The Echo Nest is HyperText Transfer Protocol (HTTP) based. This means that to retrieve information from The Echo Nest, a request is made to a Uniform Resource Locator (URL) that specifies what information is required. The result can be returned as either JavaScript Object Notation (JSON) or Extensible Markup Language (XML) depending on the parameters.

The API offers a lot of functionality; however the only function used by our application is the APIs *search* function. We use this to find information about a music track given the artist and title. If, for example, we want to find information about the song *Karma Police* by artist *Radiohead*, we would make a request such as this:

```
http://developer.echonest.com/api/v4/song/search?&artist=radiohead&title=karma%20police
```

The Echo Nest will then send a response. The format of this response can either be as JSON or XML, depending on the option chosen in the request. We have chosen to use the XML format as .Net has better support for it than JSON.

The Echo Nest is *rate limited*, meaning that it monitors how many requests are made by a single application and stops responding if too many requests are made. This means that an *API key* is supplied in the request (omitted for brevity in the example above). A free account is limited to 20 requests per minute, but it is possible to increase the number of requests with the free account to 120 per minute if the API is being used for educational purposes. To limit the amount of API calls our application makes and as a possible speedup, we decided to cache the results from The Echo Nest using a database.

All these considerations form the basis of the design of our Echo Nest library. The library is divided up into several classes that are described in the following paragraphs.

Echonest.TrackSummary

Our C# representation of what The Echo Nest calls a *audio summary*, which contains information about a audio track. This information can be attributes like loudness, duration, etc. In this application, we are only interested in the *tempo* (the number of beats per minute), the *energy* (a measure of how intense a piece of music is) of the track, and a possible URL of the album cover for the track. Hence, the `TrackSummary` only contain these three values.

Echonest.Limiter

The `Limiter` is a class responsible for delaying requests to a web API (in our case The Echo Nest) so that it keeps the number of requests below a certain threshold. It does this by keeping track of when the last request was made and ensures that the minimum delay between requests correspond to the allowed quota. If for example a maximum of 20 requests are allowed each minute, the Limiter will ensure there is a delay of at least 3 seconds between each request.

Echonest.IEchonestService

The `IEchonestService` is an interface that contains a single method that can retrieve a `TrackSummary` from an Echo Nest database. The interface is displayed in listing 1.

Listing 1: `Echonest.IEchonestService`

```
public interface IEchonestService
{
    Task<TrackSummary> GetSummary(string artist, string title);
}
```

The `GetSummary` method takes an artist and a title and returns a `TrackSummary` if it finds a matching music track. The return type is `Task<TrackSummary>` because the program uses .Net's task-based parallelism[3]. This means that instead of blocking further execution of the program until The Echo Nest answers (synchronous execution), we can do other tasks in the same thread without blocking it (asynchronous execution). Not blocking the thread is especially important if you have a graphical user interface, as blocking the thread can make the interface unresponsive.

Echonest.EchonestService

The `EchonestService` is an implementation of `Echonest.IEchonestService` using HTTP to communicate with The Echo Nest API. The `EchonestService` sends a request to The Echo Nest, parses the XML response, and returns a `TrackSummary` as a result.

The `EchonestService` contains the `LoadXDocument` method which takes care of the HTTP communication with The Echo Nest. This is done as seen in listing 2, where the code uses the specified URL to contact The Echo Nest service and request the desired information. The requested information, if found, is returned as a XML file.

Listing 2: Echonest.EchonestService.LoadXDocument

```
private async Task<XDocument> LoadXDocument(string url)
{
  HttpWebRequest request = WebRequest.CreateHttp(url);
  while (true)
  {
    await limiter.Delay();
    using (HttpWebResponse response = (HttpWebResponse)await
       request.GetResponseAsync().ConfigureAwait(false))
    {
      string limit = response.Headers["X-Ratelimit-Remaining"];
      if (limit != null)
        limiter.Limit = int.Parse(limit) - 1;

      if (response.StatusCode == HttpStatusCode.OK)
        return XDocument.Load(response.GetResponseStream());
      else if ((int)response.StatusCode == 429)
        continue; //We have sent to many requests, delay and try
           again
      else
        throw new HttpException((int)response.StatusCode,
           "Unexpected response code");
    }
  }
}
```

Another important method in the `EchonestService` class is the `GetSummary` implementation seen in listing 3. This method creates a URL to be used with the `LoadXDocument` method from the input artist and title values. The method also handles the parsing of the returned XML document into a `TrackSummary` containing the desired information.

Listing 3: Echonest.EchonestService.GetSummary

```
public async Task<TrackSummary> GetSummary(string artist, string
    title)
{
  if (artist == null)
    throw new ArgumentNullException("artist");
  if (title == null)
    throw new ArgumentNullException("title");

  string request
     ="http://developer.echonest.com/api/v4/song/search?" +
    "format=xml&results=1&bucket=audio_summary" +
    "&bucket=id:7digital-US&bucket=tracks" +
    "&api_key=" + HttpUtility.UrlEncode(Key) +
    "&title=" + HttpUtility.UrlEncode(title) +
    "&artist=" + HttpUtility.UrlEncode(artist);

  XDocument doc = await LoadXDocument(request);
  XElement song =
     doc.Element("response").Element("songs").Element("song");

  if (song == null)
    return null;

  XElement track = song.Element("tracks").Element("track");
  XElement summary = song.Element("audio_summary");

  if (summary == null)
    return null;

  return new TrackSummary
  {
    Energy = Double.Parse(summary.Element("energy").Value,
       CultureInfo.InvariantCulture),
    Tempo = Double.Parse(summary.Element("tempo").Value,
       CultureInfo.InvariantCulture),
    CoverURL = track == null ? null :
       track.Element("release_image").Value
  };
}
```

Echonest.Cache.EchonestCache

The `EchonestCache` is also an implementation of the `Echonest.IEchonestService` and creates a caching layer over another `IEchonestService`. The requests are cached to the disk using a database (see section 9.2) to eliminate the need to contact The Echo Nest API to get song-information every time the program is restarted.

This can be used with `Echonest.EchonestService` to create an efficient dynamic caching system. Since the interface `Echonest.IEchonestService` is implemented for both, the application does not have to operate in any different way if it is cached or not. An example of this can be seen in listing 4, where an `EchonestCache` is created over an `EchonestService` and a request is made.

Listing 4: Example usage of `EchonestCache`

```
using (EchonestCache cache = new EchonestCache("database.sdf",
                             new EchonestService(key, 100)))
    Console.WriteLine(cache.GetSummary("Radiohead", "Karma Police"));
```

Prototype.Exercise.ExercisePlan

The `ExercisePlan` is a representation of an exercise plan that the user can start. It is a single class that includes a name, a warm-up duration, a cool-down duration, a low intensity interval duration, a high intensity interval duration and the number of interval repetitions. This model is accurately able to represent simple exercise plans, but it is not suitable for more advanced plans. The reason for this simple model being chosen is the focus of this project being to get the core functionality working properly, with flexibility not being as much of a concern. A more flexible approach would be to represent the exercise plan as a class containing several sets of intervals representing the separate parts of the plan. This model might be used in a final solution if one were to be built.

Prototype.Media

Prototype.Media is made up of the three classes `Jukebox`, `M3UReader`, and `TagLibOperations` as well as the interface `ISongGetter`.

The `Jukebox` class implements the `ISongGetter`, which includes a single method called `GetSongWithIntensity` which returns the file name of a track that fits the specified intensity. The `Jukebox` also implements the k-means algorithm, described in chapter 19, as a 3-means to sort the songs form a playlist into the three intensity groups after having consulted an `IEchonestService` to get the tempos of the tracks.

The `M3UReader` class is static and contains a single method called `ReadFromFile`, which returns a collection of file names from a playlist file of the .m3u file format.

The `TagLibOperations` class is also static and contains the methods `GetArtist`, `GetTitle` and `GetAlbum` which reads and returns, if possible, the artist, title, or album title from the file's metadata.

9.2. Databases

To cache data about the individual tracks, retrieved form The Echo Nest, a database structure is required. Databases are used to store data in a secure way by following the ACID (Atomicity, Consistency, Isolation and Durability) properties which ensures that data is consistent and that changes on data points affect only the targeted data.

For this project SQL Server Compact (SQLCE) is used, SQLCE is a Microsoft SQL Server (MSSQL) variant that is available for free and can be used as an embedded part of a C# application. With SQLCE it is possible to deploy the SQLCE service directly with the developed desktop or web application. The SQLCE service supports the same SQL syntax as other editions of SQL Server, which makes it easier to migrate the database to another SQL Server edition[18].

The database should store the information about a track according to the combination of artist and title, as several tracks can have the same name or the same artist. Even though a track might have the same title or artist as another track, a track cannot have both the same artist and the same title as any other track without being the same song. Here we can use a composite key feature, which requires the combination of the values marked as composite key (in our application the artist and title) to be unique in the table.

The database will be used as a cache for The Echo Nest service described in section 9.1, which means that if information is requested for a track, the system will first check the cache before making a request to The Echo Nest. If the data is not stored in the cache the system will try to obtain it from The Echo Nest service and thereafter store it in the cache for the next time the information is needed. The process of getting the wanted information for a specific song is visualized in figure 9.1.

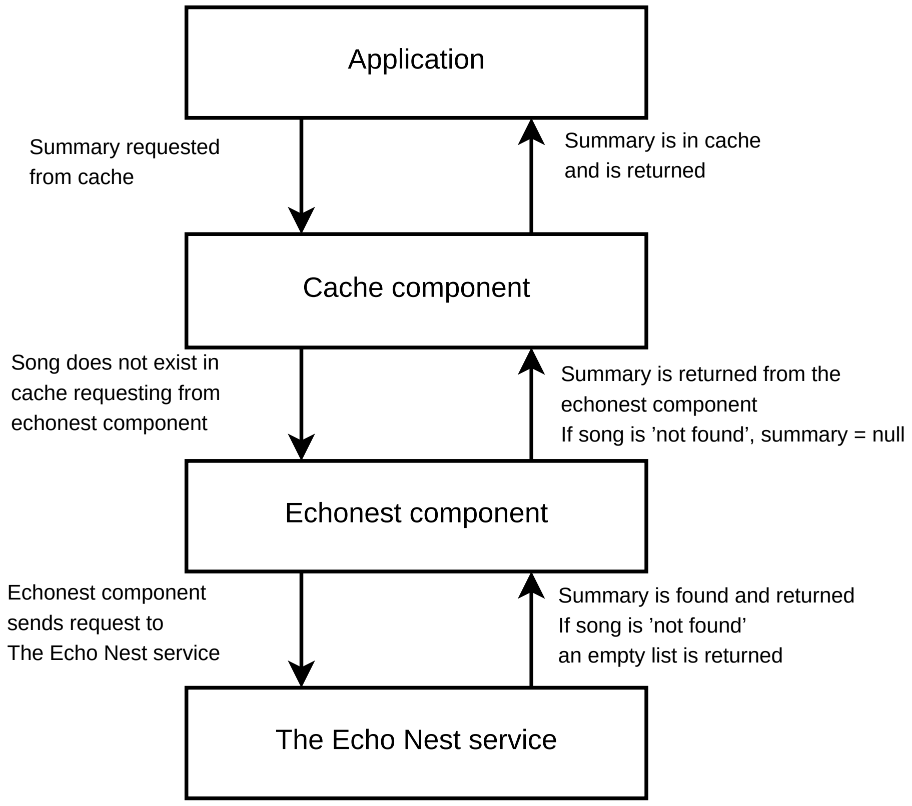

Application
Summary requested from cache
Summary is in cache and is returned
Cache component
Song does not exist in cache requesting from echonest component
Summary is returned from the echonest component If song is 'not found', summary = null
Echonest component
Echonest component sends request to The Echo Nest service
Summary is found and returned If song is 'not found' an empty list is returned
The Echo Nest service

Figure 9.1.: Echo Nest Cache

9.3. TagLib#

TagLib# is a free .NET library for reading and writing metadata for music files. The library is released under the LPGL license, which means it can be freely used by anyone, even for proprietary applications. In this project, TagLib# will only be used to read the metadata from the music files in order to get the artist and title needed for analyzing the track with The Echo Nest service. TagLib# is fast, easy to use, and lightweight making it ideal for this project.

When the application is instructed to open a playlist file to use with an exercise plan, the music tracks contained in the playlist are opened with TagLib# and the individual track's title and artist is retrieved from the music file's metadata container. The program will then use this information with The Echo Nest service and get the tempo, as well as other data, for the individual tracks. A list of the tracks is then produced, using the k-means algorithm described in chapter 19, categorized after tempo and can now be used to get any of the playlist's songs that match a certain intensity.

The user interface also uses the TagLib# library to display various information to the user about the currently playing track. This information could be, but is not restricted to, title, artist, album title, album cover art, genre, and year of release as long as this information is availbale from the music file's metadata container. If the music file does not contain an album cover art, The Echo Nest service might be able to find an URL linking to the official cover art of that track/album.

9.4. Song selection

To select a song from a playlist with a desired intensity, the tracks from the playlist will be divided into three groups, each representing an intensity level. There are three levels of intensity; low, medium and high intensity. Dividing the user's songs into exactly three groups provides a group of low intensity songs to use while the user is warming up or cooling down, a group of songs for when the user is running at the low intensity interval, and a group of songs for when the user is running at a high intensity interval. Defining an intensity level for each song before starting an exercise session eliminates the need to analyze music while the exerciser is running as this could lead to unwanted delays due to the asynchronous request-response model used for analyzing the music with The Echo Nest. Selecting a song that fits a specific interval in the exercise plan is reduced to selecting a song from the corresponding group/category that contains all of the songs for that intensity.

The definition of the intensity level of a certain song could be implemented in different ways. One way is to group the songs by their tempo, giving the user a steady beat to run to. However some songs may be in a “gray zone” between two intervals as seen in figure 9.2 where the user might not be able to determine what group the currently playing song belongs to. The factor to push the track out of a “gray zone” could be the energy of the track and this data is also available from The Echo Nest. Energy describes how energetic a song is and ranges from 0 to 1. The benefits of this grouping method is that the user can decide for himself/herself where to split between low, medium and high intensity. The disadvantages is that a playlist could potentially not contain any song within the tempo-interval defining a specific intensity, and could therefore possibly not contain any songs of e.g. low intensity.

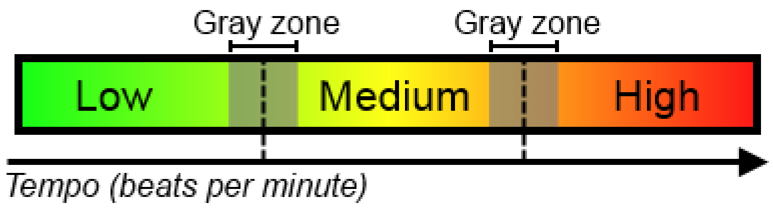


Figure 9.2.: Intensity groups and their gray zones

Another method of grouping the songs into intervals of intensities is to dynamically determine what tempos that should define the split between low/medium and medium/high intensity. This method would require a more advanced algorithm but will eliminate the possibility of a playlist creating an empty interval. The disadvantage with this method is that the user cannot choose the tempo that defines the low, medium and high intensity intervals. This method also does not ensure that songs with “high” tempo will be placed in the high intensity group if all the songs in the playlist have a “high” tempo. This is the method chosen for the application and is implemented as a clustering algorithm. This algorithm, as well as its limitations, is described in chapter 19.

10. Design of Graphical User Interface

This chapter will contain considerations taken into account during the development of the prototype's graphical user interface (GUI).

10.1. Design principles

To ensure the best user experience for the exercisers using the prototype, the GUI will be formed around the twelve design principles described in the textbook for the *Design and Evaluation of User Interfaces (DEB)* course [6]. In addition, some of the Android Design Standards[1] has been followed as well while designing the user interface for the application.

Learnability

The first four principles tries to explain what to do to the make the program easy to use and remembering how to use it.

1. Visibility - The user needs to see what the system is currently doing and what functions are available. It's much easier for a user to recognize elements rather than recall them. The visibility principle plays a very important role in a user's first impression of the system.

2. Consistency - A consistent design throughout the system makes it easier for a user to learn to use the system. Furthermore, a system with a design consistent with that of other systems will take advantage of the user's experience with those other systems and increase the ease of learning.

3. Familiarity - Choice of language and symbols have to be understandable and well known by the users. This influences how fast the user learns to navigate the system.

4. Affordance - Buttons needs to afford the users to press them. The same goes for textboxes, lists, and so on. These designs should match that of other systems the user is already familiar with.

Effectiveness

The next five principles tries to explain what to do to ease the use of the system and how to assist the user in case of errors.

5. Navigation - The users need to be able to move around between the different parts of the program in an intuitive way.

6. Control - Make sure who or what is in control of the system. Users should have the needed controls to use the system. Also, if the system is waiting for response from the user, it should make the user aware of it.

7. Feedback - The user needs constant feedback for the actions and choices that the user makes. The feedback has to be consistent to reduce possible confusion for the user.

8. Recovery - Recovery functions or messages have to be enabled in case of errors or mistakes. The recovery functions or messages have to be quick and effective.

9. Constraints - To prevent mistakes or errors the users should be constrained from doing actions in a wrong order and from giving the system invalid input.

Accommodation

These last three principles tries to explain what to do to deal with accommodating different users and respecting those differences.

10. Flexibility - The system have to provide several ways to execute operations to support several different users with different levels of experience with the system.

11. Style - The design have to be stylish and attractive to invite the user to use the system and should not be a visual annoyance to the user (e.g. misaligned controls and too many complimentary colors).

12. Conviviality - The system have to be polite and friendly in the feedback to the user. The user needs to be invited to use the system, not offended by it.

10.2. Graphical User Interface navigation

This section will describe how the GUI is directly implemented in the prototype, references to the design principles above looks like this: (supports the 5th design principle ***navigation***).

In section 4.3 the User Interface (UI) is described and the UI navigation diagram is illustrated in figure 4.2. The GUI navigation diagram is shown in figure 10.1 and is a simplified diagram of the UI navigation diagram. As the GUI navigation diagram illustrates, the GUI consist of five screens and a *LoadingScreen* that the user can interact with while using the prototype.

Besides the *LoadingScreen*, which is an "overlay screen"[1], and the *SelectPlaylistScreen* the GUI consist of a *StartScreen, an ExercisePlanPickerScreen, a CreateExercisePlan-Screen,* and a *PlayScreen.* The *StartScreen* is the first screen the user will meet when starting the prototype.

As shown in the navigation diagram in figure 10.1 the user can from the *StartScreen* get to the *ExercisePlanPickerScreen* and the *CreateExercisePlanScreen.* The *StartScreen* (see figure 10.2a) acts as a "main menu" with some shortcuts to the most used functions of the program which enhances the 5th design principle; **navigation**.

[1]An overlay screen is a screen that runs on top of the previous with some kind of feedback to the user.

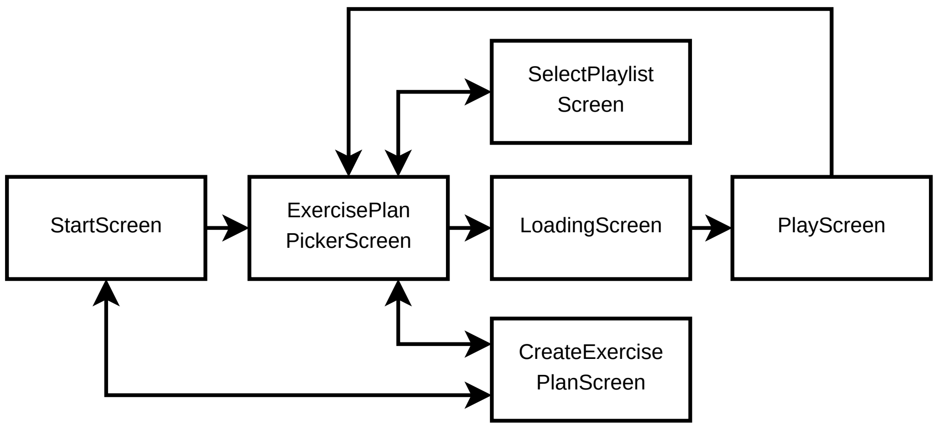


Figure 10.1.: Navigation diagram of the prototypes Graphical User Interface

The *ExercisePlanPickerScreen* illustrated in figure 10.2b is one of the three important screens of the prototype. Here the user can choose a playlist and an exercise plan to start a new exercise session, advancing the program to the *PlayScreen*.

Because loading time may occur between the *ExercisePlanPickerScreen* and the *PlayScreen*, as the music is analyzed using The Echo Nest service, the screens can be separated by an overlaid *LoadingScreen* (in the working prototype a basic Windows loading circle appears in place of the cursor) to support the 1st and the 7th design principles; **visibility** and **feedback**. This is done to show the user that the system is busy and needs some time before the user can continue.

To advance to the *PlayScreen* the user needs to choose a playlist. This action varies between operation systems. In the working prototype a basic Windows "Open file" menu appears. An different, but equivalent, option will be available on other operating systems such as Android or iOS.

10.3. Usability

The initial usability of the system is based on some of the standards defined by the And [1] and also on this semesters DEB course. The usability of a system is measured by the efficiency and understandability of the methods that the system uses to interact with the user. This interaction can be manifested through text, buttons, screens, images, instructions, feedback, and other controls in the interfaces between the system and the user. The following paragraphs will describe the considerations made for each screen in the prototype in relation to the usability of that screen.

10.3.1. *StartScreen*

The *StartScreen* (see figure 10.2a) is the first screen the user meets and it is therefore very light in its design. It has a title in a large font, a program icon image, and three main buttons as well as a button for changing the language of the program. The three main buttons stretch to fill the screen horizontally making room for longer messages on the buttons. As the buttons all lead to new screens or exits the program they are labeled with a necessarily detailed text.

This screen supports 3th design principle **familiarity**, by switching language matching the operations systems language, for now the prototype support two language; Danish and English.

Pressing the "Exit" button prompts the user to confirm the action to avoid the inconvenience if the user accidentally clicked it.

10.3.2. *ExercisePlanPickerScreen*

The *ExercisePlanPickerScreen* (see figure 10.2b) is one of the screens that a user can navigate to from the main menu. As it is a subscreen and defines a main functionality of the system it has a title. The title is shown at the top of the screen and describes the screens functionality. If the user is in doubt of the currently visible screen he/she would look at the top of the screen for the answer (supports the 1th design principle **visibility**).

The rest of the screen is separated into three rows by horizontal lines. The top row contains controls for manipulating the chosen playlist. This includes two labels and a button. The top label shows the static text "Playlist:" indicating that this row is related to choosing a playlist. The button prompts for a new playlist and is therefore labeled as "Change". The bottom label displays the name of the currently chosen playlist or "none" if none is chosen. This label is the feedback from the "Change" button signaling the user that the system has accepted the user's input (supports the 6th design principle **control**).

The second row contains a title, a list, and three buttons. The title gives the user a task; "Choose an exercise plan:" which is a necessary task to complete before the user can continue in the program. The list contains the available exercise plans and only one can be selected at any time. The three buttons; "Add", "Edit", and "Delete" defines actions involving exercise plans and is therefore grouped with the list containing the exercise plans to make the user associate the buttons actions with the plans. Even though the images on the buttons should be enough for a user to identify what action lies behind it, a label was added below each button with a short description of what each buttons action was. As editing and deleting is not valid actions with no plan selected, the two corresponding buttons are disabled until the user selects a plan in the list (supports the 9th design principle **constraints**).

The last row contains two buttons for navigation. A “Back” button located to the left and a “Play” button on the right. The “Play” button is also first enabled when the user selects a plan in the list, these buttons are symbolized as arrows with the user can associate with other familiar system, such as browsers (this supports the 3th and 4th design principle **familiarity** and **affordance**, because the users “should know” what the buttons means and thereby be invited to use them).

10.3.3. *CreateExercisePlanScreen*

The *CreateExercisePlanScreen* (see figure 10.2c) is available from both the *StartScreen* and the *ExercisePlanPickerScreen* and serves two purposes; creating new exercise plans and editing existing plans. As with the *ExercisePlanPickerScreen* this screen has a title at the top of the screen. The title shows whether the screen was invoked with the intention of creating a new plan or editing an existing plan.

Below the title seven groupings are stacked on top of each other. Common for the first six groups is a label showing what setting the group controls. The first control group contains only the label “Name” and a textbox. This is understandable by most users and does not require any further description. The next three control groups, as well as the sixth group, contains a textbox and a slider in addition to the label. These groups control the duration of an interval or period in the plan. The slider and the textbox provide two different methods of changing this value. As they both control the same variable they are linked so that altering the value of one of the controls, be it the textbox or the slider, the other control changes to the corresponding value. The slider would be preferred on a device with a touch screen as the default method of input and changes the setting with ten seconds per tick. The textbox is provided for precision input (supports the 10th design principle **flexibility**, by letting different users have different ways to manage an exercise plan).

The fifth control group contains two buttons separated by a textbox in addition to the description label. They control the value of an integer. The two buttons decrement and increment the value and are labeled with a “-” and “+” sign. The textbox provides a manual method of input and also shows the currently selected value, and this textbox’s value is limited to positive integers above zero to avoid errors (supports the 9th design principle **constraints**).

The bottom group contains two buttons; a “Save” button and a “Cancel” button. Pressing the “Cancel” button prompts the user to confirm the action to avoid loss of data if the user accidentally clicked it.

10.3.4. *PlayScreen*

The *PlayScreen* (see figure 10.2d) also contains a title at the top of the screen to inform the user of the screen’s purpose. The three possibilities of interaction with this screen is grouped at the bottom of the screen.

These actions are; “Stop”, “Play/Pause”, and “Skip”. The buttons actions are only described with the image that they show. As they are the standard symbols for media control the user should know what each button does (supports the 3th and 4th design principle **familiarity** and **affordance**). Because the “Play/Pause” has two functions it changes icon depending on the possible action of the button. The “Skip” button is also only enabled when the media is playing (supports the 9th design principle **constraints**). The “Stop” button takes the user back to the previous screen and stops the exercise session.

Above the buttons two labels show the artist and the title of the music track currently playing. Above that the user can see the current intensity level (low, medium, or high) in the exercise plan as well as the corresponding color (green, yellow, or red). They are color coded so that the user does not have to read text while exercising to determine what the current intensity is. In the middle of the screen the album cover art for the currently playing track is shown if possible to reassure the user that it is playing the correct track.

10.3.5. Error-handling

In some use cases the user could potentially make errors. When the user does so, he/she needs to be informed about it and be instructed in how to correct the error. Doing so will enforce the 7th and 8th design principles **feedback** and **recovery**. The prototype will catch the user’s attention with pop-up windows taking focus from the main window. The user will not be able to continue with the program until the pop-up window has been closed making sure that the user is notified of the error. The user is constrained from making other errors than described here.

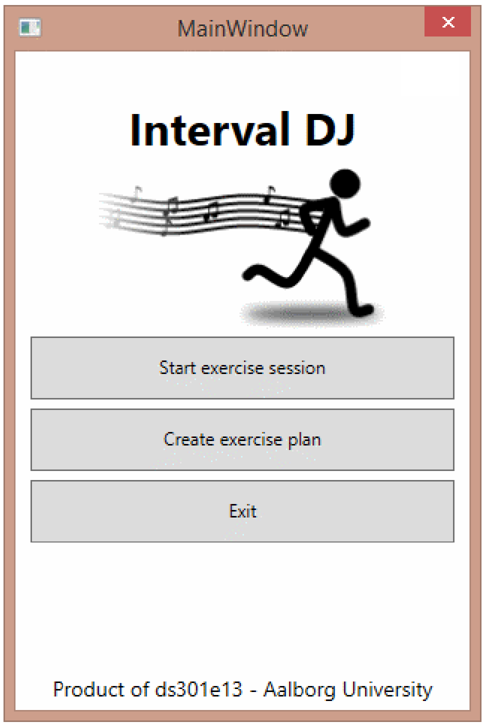
When the user creates a new exercise plan or edits an existing one, several constraints exist on the name of the plan. It should not contain only whitespace as that could potentially confuse the user later and it must be unique. It must be unique because the database is storing each plan with the name as the primary key, but having plans with the same name could also confuse the user. However, the program only checks for a valid name when the user clicks the “save” button. If the name is not valid at that time, the plan cannot be saved and the user needs to be informed of this. The program will tell the user that the chosen name is not valid and the reason for why it is not valid. Telling the user why the name is not valid is essentially to tell the user how to correct the error.

When the user wants to start an exercise session, he/she needs to choose both an exercise plan and a playlist to use with the plan. If the user forgets to choose a playlist, the program will inform the user of this and explain that a playlist must be chosen before the program can proceed.

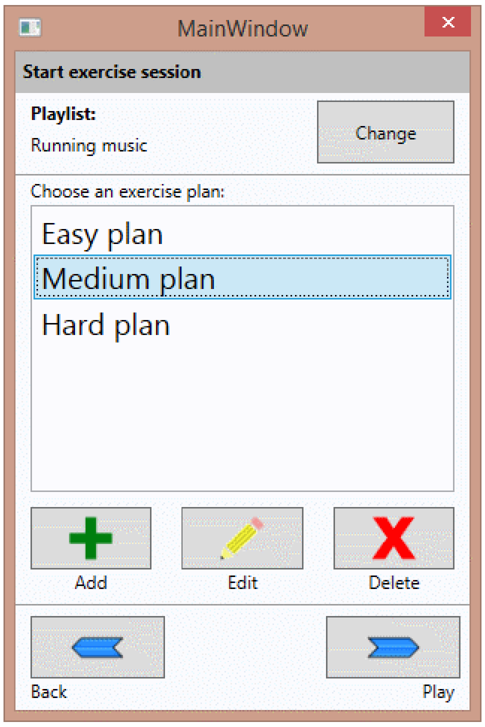
The error-messages are presented to the user in a neutral language and friendly tone, inviting the user to correct the errors him/herself instead of seeking technical assistance elsewhere (supports the 12th design principle **conviviality**). If the user forgets to pick a playlist when trying to continue the program gives the user the following message: “Please select a playlist first by clicking the "Change" button”. This tells the user that further action has to be taken to continue, and specifically directs the user in correcting the problem by telling him/her what to click.

10.3.6. Style

Generally the GUI is designed in a minimalistic-style to prevent confusion and improve the consistency throughout the system. The white background with gray buttons are consistent through the system (supports the 2nd design principle **consistency**), so the user quickly will learn that it is buttons. The style of some of the buttons is optimized with icons (see figure 10.2b). These buttons has been represented with icons as they activate well known actions from several situations (e.g. add, edit, remove, play, pause and stop) and therefore has well known icons associated with them. Buttons that represent actions that are somehow more specific for this program is not represented with icons, but instead with a description of the action (e.g. Start exercise session, create exercise plan).



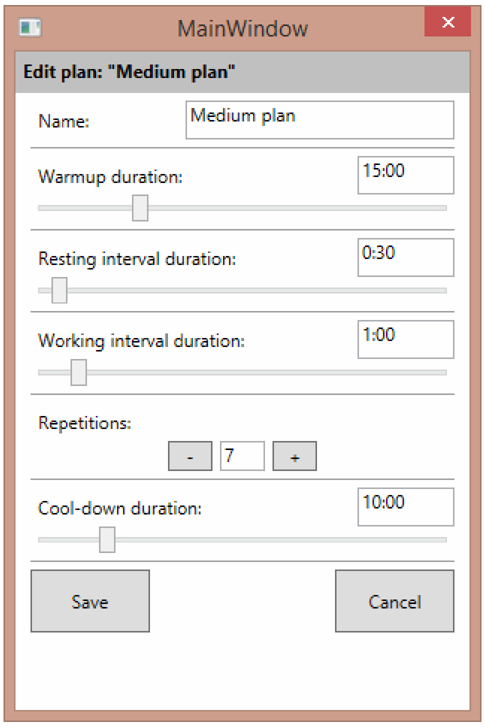
(a) *StartScreen*



Playlist:
Running music

Medium plan

(b) *ExercisePlanPickerScreen*



Edit plan: "Medium plan"

Name: Medium plan

Warmup duration: 15:00

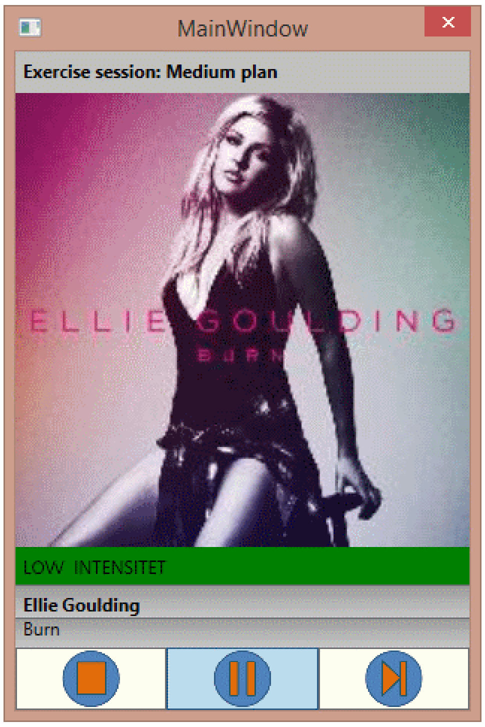
Resting interval duration: 0:30

Working interval duration: 1:00

Repetitions: - 7 +

Cool-down duration: 10:00

(c) *CreateExercisePlanScreen*



Exercise session: Medium plan

LOW INTENSITET

Ellie Goulding
Burn

(d) *PlayScreen*

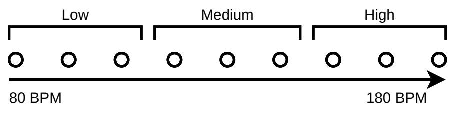
Figure 10.2.: Screenshots of the prototypes GUI

11. Clustering algorithm

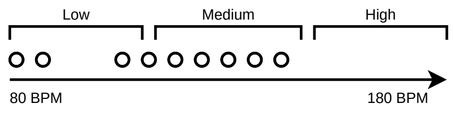
In our application, we need to partition a playlist with several music tracks into three intensity groups: low, medium and high (see section 9.4).

Several methods were considered, starting with a simple solution where a preset tempo interval was assigned to each category. Songs would then be independently categorized by their tempo. This method works well in cases where the songs are spread in a uniform distribution between the different intervals (see figure 11.1a), but it leads to problems when this is not the case, e.g. if there are no songs that fit in the high interval category (see figure 11.1b). This result is highly undesirable, as we would like to have a number of songs in each category.

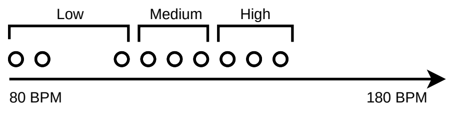
A better method would be to sort the list of songs by their tempo and then partition the lowest $1/3$ to the low category, the middle $1/3$ to the medium category, and the upper $1/3$ to the high category. This leads to better results as it ensures a close to equal amount of songs for each category (see figure 11.1c), but it still has some minor problems (see section 9.4). In figure 11.1c, we can see how the low interval is stretched to include one more song. It would be better for the categories to remain distinct, so that the transition from one category to another can be clearly heard in the change of tempo. It would therefore be better to have a kind of clustering behavior as seen in figure 11.1d.



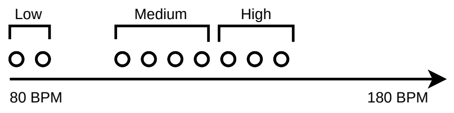
(a) Ideal uniform tempo distribution



(b) Nonuniform tempo distribution - no high-intensity music tracks



(c) Intervals changed to partition songs equally



(d) Clustered intervals

Figure 11.1.: Different categorization methods

To achieve this clustering behavior, it is necessary to use a *clustering algorithm*. One relatively simple and often-used clustering algorithm is the *k-means algorithm* originally described by MacQueen [15].

Since we are dealing with 1-dimensional data, there are other, more specialized, algorithms for data clustering. One of these is *Jenks natural breaks optimization*[16], which is an algorithm originally designed to for selecting ranges for Choropleth mapping (see figure 11.2). This algorithm is supposedly better than k-means for 1-dimensional data[16].

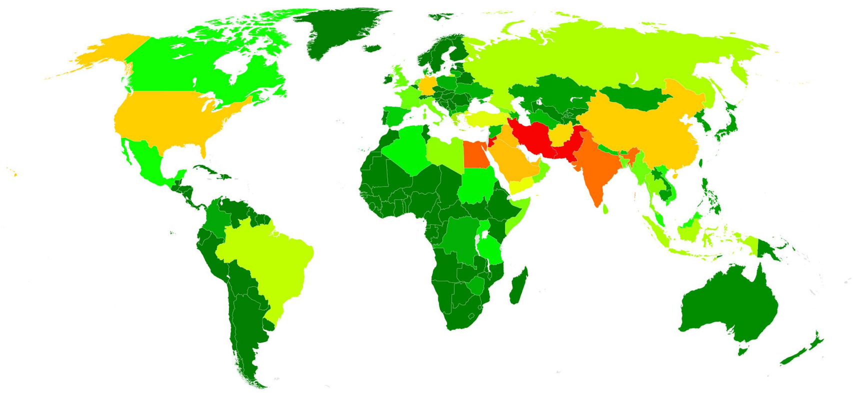


Figure 11.2.: Example of Choropleth mapping. Image source: Wikipedia[20]

The problem with Jenks natural breaks optimization lies primarily in its lack of documentation. The primary source in which the algorithm was originally described is seemingly not available either online or in print[16]. There are implementations of the Jenks natural breaks optimization, but these are mostly ports from other implementations. Thus, they only describe what the algorithm does, not how the algorithms works.

Due to these problems, we have chosen to proceed with the k-means algorithm even though Jenks natural breaks optimization might be better. The k-means algorithm is described in more detail in chapter 19.

12. Recommendations

This chapter will shortly recap the design part as well as describe the thoughts made for the initial use and implementation.

12.1. The utility of the system

The system design meets the most essential demands from our analysis, however the system's usefulness as well as correctness and understandability can only be known when the first prototype have been developed. As explained earlier the prototype will be developed for a PC platform but should be fairly easy to port to other mobile platforms if needed. This will however require several additional resources. With the current stage of development and time left for the project the only realizable prototype is the PC prototype. It is recommended that this prototype should be developed as a complete functional prototype containing most of the described features from the design specifications. It should furthermore be used for usability testing and should therefore contain a fairly complete graphical user interface.

12.2. Plan for initiating use

We expect the users of the system to have some basic knowledge about modern mobile applications[1] and how to use them. The system should implement a help function, allowing users to learn how to use the program based on a basic knowledge about modern mobile applications if there is time. Also having the user interface translated to different languages would be preferred. The final version of the system would be maintained through some of the popular mobile application stores, where updates would be provided, as well as the handling of bug reports for post-developing maintenance.

[1]Applications for smartphones or other devices with a touch screen interface.

12.3. Implementation plan

The PC prototype development will be led by one or two lead programmers who will be assisted by the remaining members of the group. The prototype is expected to be ready for functional and usability tests in about 4-6 weeks. The testing of the prototype should not take any more than two weeks including both conducting the tests and analyzing the results. There will probably not be any time to develop a second prototype based on the test results from the PC prototype as there is currently only 9 weeks left of the project time.

Part III.

Test and Reflection

13. Functional test

Testing is a very important part of programming, as it provides a way to ensure that the program code does what it is designed to do.

While developing the program most of the key functions were manually tested as they were developed and changed. This resulted in the program being tested continually, to see if the different functionality in the program worked as expected. If the program did not behave as expected (e.g. throwing an exception, giving strange results or just looked wrong) the program would be debugged using the Visual Studio debugging tool and the code would be reviewed to see where the error was located. The program was tested according to the test plan (see appendix B). This test plan includes information on the approach and tasks used for the test as well as the features and items tested.

14. Usability test

This chapter will focus on the usability testing of the developed prototype. The usability testing involved several participants who were asked to perform several tasks. The reason of the test is to verify that the quality of the graphical user interface is at a level of usability such that the program is usable by a broad range of people. To define and validate the purpose of the test a test plan was made.

14.1. Test plan

The purpose of the test plan is to describe the scope, used resources, and schedule of the test as well as the method of approach. The test plan will also describe the features to test and how they will be tested.

14.1.1. Features to be tested

This test will test the participants ability to navigate between the screens in the graphical user interface of the developed prototype as well as their ability to use its functions without assistance. This will show whether the program is intuitive and easy to understand and learn, or whether it is confusing and hard to understand. The features that could be tested are; creating, editing, and deleting exercise plans, choosing a playlist to use, and starting an exercise session. As some of these features are trivial and less important this test plan will focus on describing how to test the two main features; creating an exercise plan and starting an exercise session.

14.1.2. Approach

As the purpose of this test is to evaluate the usability of the developed prototype, the tasks to put each participant through should be based on the major use cases of the system. This will ensure that the intended use of the system is tested and evaluated as individual parts of a whole.

As the system is aimed at using the user's music and every participant cannot be asked to bring their music to the test sessions, a standard playlist should be available for each participant during the test (as the program requires music to function). This will also ensure that every test session will take place in the exact same environment making the test take place in a controlled setting.

The participants will be asked to go through two test tasks alone with the oversight of a test conductor (which will act as a lifeline/helper) and a logger (which will take notes throughout the test). The sound and actions taken by the participant will be recorded with a screen-video capture program running on the machine hosting the test environment and used for further analysis. After each session the participants should be asked a set of questions to get a view of the participant's experience as a whole.

14.1.3. Success/failure criteria

A test task will be defined as successful if the participant is able to complete the task within the target time-frame and without receiving help from the test conductor. If the user surpasses the maximum amount of time allocated to the task or otherwise gets stuck or lost the task will be defined as failed, and extra resources should be used to determine why the participant failed at the task.

14.1.4. Testing tasks

To setup the test one computer must be running the version of the prototype to be tested. The computer should either have access to the Internet or have pre-cached the necessary data to use the program. Before each test session the in-program actions of the previous participant should be removed by restarting the program and replacing the database with the pre-made database containing the necessary data for the test cases. The screen-capture software should also be reinitialized to record a new video before the next participant starts his/her test session. During the test each participant will be asked to execute the following tasks:

- Create a new exercise plan with some specific parameters (the task should specify the parameters)
- Start a specific exercise plan then skip a song and terminate the program (the task should define the name of the plan to start)

After each task has been completed or failed, as defined by the success/failure criteria in the previous section, the participant should be able to give feedback by answering some questions prepared by the test conductor (these questions can be seen in subsection 14.2.2). The concrete tasks that each participant was asked to complete can be seen in appendix E.

14.1.5. Schedule

The individual test sessions will consist of a setup/resetting period followed by a testing period where the participant is asked to perform the tasks defined in the testing tasks section. After the session the participant will be interviewed/take a survey. Prior to the test sessions the test equipment should be set up and each test case should be tested by the test conductor(s) to verify that they work as intended. A whole session worth of time should be assigned to the initial setup and verification of the testing equipment.

Setting up the test equipment and verifying that it works should not take more than half an hour. Resetting the database in between tests is relatively easy and should not take more than half a minute (it could even be automated with a script significantly reducing the needed time), and resetting the screen recording software would increase the total resetting time to roughly a minute. Introducing each test participant to the program and acquiring their consent of participation would also take a minute. The maximum time for the first task has been set to 2 minutes and 30 seconds. If any participant has not completed the first task within 2 minutes and 30 seconds the task should be declared failed. The second task should not take any more than 1 minute and if it does that test case will likewise be declared failed. Interviewing the participant after the tasks has been completed/failed would not take any more than 2 minutes and could possibly be done while the next participant tests the program.

This sums to a total test-time of approximately 6-7 minutes per participant which allows for roughly 5-6 test participants per hour.

14.2. Preparation

14.2.1. Recruiting participants

Potential participants for the prototype could be anyone in theory. But knowing some of the jargon that the program uses would definitely make the testing process more effective. This is why participants for the usability testing primarily were searched for in the nearby running clubs/associations. The recruiting was initially aimed at finding a good spread between the participants ages and close to an equal amount of male and female participants.

Due to the time passing on and the disagreement of the potential test-group to schedule a test, we decided to change the target group in a drastic way. We contacted Friis Aalborg (a shopping center located in central Aalborg) and were allowed to test our systems for two hours on their visiting customers if they voluntarily agreed to participate in the test. In the end we tested 10 shoppers within the age of 20-50 where 5 participants were male and 5 were female.

14.2.2. Closing questions

After the completion of the usability test each participant was asked if he/she had any trouble using any part of the software and if so if the user cared to explain what their thoughts on the problem was and how they experienced it. They were also asked what kind of previous experience they had with the use of IT systems to determine how the experienced problems would be interpreted by the user in terms of severity.

14.3. Execution

The test team met at Friis Aalborg at the agreed time and contacted the guard who showed the way to the designated place where the usability test were to be conducted. The team had brought a small stand with room for a laptop computer and a wireless mouse. The computer was setup with the application with nothing else on the screen to distract the user. The test wes conducted at a relatively quiet spot which resulted in a testing environment with as little background noise as possible in the given situation.

Two members of the team asked individuals passing by if they would like to participate in the usability test. Only people above the age of 20 were asked. Whenever a user agreed to participate in the usability test, he/she would be informed of the project and of the developed application. The user would then be asked to sign an agreement, stating that they accepted that the test results could be used in the project report and that they would remain anonymous. This agreement can be seen in appendix F. Hereafter the user would be handed a piece of paper containing the two tasks for the user to complete. These tasks can be seen in appendix E.

The test would be completely conducted by the user with the assistance of the test conductor. The conductor would introduce the user to the program and provide him/her with the tasks and only intervene if absolutely crucial for the continuation of the test or if the user would use too long time completing a task or would become stuck. A member of the group was also assigned to take notes on how the user performed and reacted during the test, how much time was used on each task and what the user seemed to have trouble doing. The users mouse-interaction with the application as well as the users verbal comments while using the application would be recorded using the open source software CamStudio, which is a piece of software used to capture a video of onscreen activities as well as audio, either from the computer's speakers (stereomix) or a microphone.

14.4. Conclusion/summary

14.4.1. Identified problems

After the test had been carried out the following usability problems were identified and are described below. Some of the more concrete problems also contains some possible solutions for solving/suppressing the problem.

(P1) Users thought that the "name" textbox when creating a new exercise plan referred to their own name. The "name" textbox in the create exercise screen (see figure 10.2c) refers to the name of the plan, not the name of the user. In our test, users frequently (in 5/10 cases) entered their own name as the name of the plan, even though our task explicitly stated how they should name the plan (see appendix E). We believe that this problem is influenced by the testing environment - the user might think that we had to use their name for research purposes, so this problem might be less prevalent in a more normal use case. One solution to this problem could be to change the "name" label to "plan name" instead to remove the ambiguity.

(P2) Ambiguous Danish labels. The Danish translation of the program used "længde af" for "duration of". This is not incorrect per se, but it creates an ambiguity as "længde af" can mean both extent through time and extent through space. So "længde af opvarming" (duration of warm up) can be understood as both the distance and the duration of the warm up. The textboxes hints to it being duration by using the format 0:00, but this might not be clear enough. Only one user noticed this problem, so it is considered a low-priority cosmetic concern. One way of fixing this could be to use the more uncommon word "varighed" instead of "længde", as it avoids these ambiguities.

(P3) Users did not notice the "name" textbox. Many users did not notice the "name" textbox until they tried to click "save" and received an error message. Many users would then backtrack up the create exercise screen and find the "name" textbox. We believe this is mainly caused by the fact that the textbox and the background is white, making the only visible part of the textbox a gray border and the "name" label. Changing either the background or the textbox to a darker color might be a solution to this problem.

(P4) Users had problems using sliders to adjust duration of exercise. Many of the users exclusively used the slider-controls in the "create new exercise" screen to set the duration settings of the plan. Almost all of these participants also took a lot of time setting the sliders to the correct position trying to hit the exact target value while bouncing back and forth between just above the target value to just below the value. All of the users encountering this problem expressed their frustration with the sensitivity of either the slider or the mouse/touchpad during the test or after the test while being interviewed. A couple of users struggling with the sliders found that they could manually input the duration by clicking the textboxes showing the duration of the setting. Those who found that they could do this also ignored all following sliders and instead used the textboxes for changing the duration settings.

Part of this problem might be due to the sensitivity of the wireless mouse used for the test, as participants that used the touchpad instead seemed to struggle less with the sliders but the sensitivity of the sliders did seem to cause some annoyance to the users and maybe needs to be lowered. And alternative would be to make sure the user knows that it is possible to use the textbox instead of the sliders as some of the participants discovered.

The problem only seemed to be present in the case of the warm-up and cool-down sliders and almost non-existent with the high and low interval duration sliders. This might be due to the warm-up and cool-down sliders having a lot more possible positions for the slider to be in, which in turn requires more precision by the user if a specific duration in wanted. Lowering the number of possible positions would reduce the severity of this problem.

(P5) Users had difficulties navigating to the play screen. Several users experienced that navigating to the play screen from the exercise-picker screen was difficult. When meeting this problem the users reacted in different ways. Some clicked the wrong button ("change playlist" button or "add plan" button labeled "Change" and "add" respectively and the "add plan" button being represented with a green addition symbol) and others even tried to ask for help. Some users just paused for quite a while before pressing the correct button to continue. All of the users that encountered this problem eventually found the right button within a time-frame of about 5-20 seconds by pausing at the screen and looking at all the options available. This might indicate that too much information and/or too many actions were available to the user at this screen as one of the participants noted when interviewed after the test.

(P6) Users pressed "Change playlist" when trying to get to the play screen. As described in problem 5 some users pressed the wrong button to try to progress to the play screen in the program. One user pressed the "change playlist" button and others moved the cursor to the button after selecting the instructed plan but did not press it after further inspection of its context. For some reason this button seemed to attract more attention than the three buttons in the bottom of the screen that becomes active when a user selects an exercise plan from the list (one of these three buttons being the correct button for advancing to the play screen).

It might be that the change from disabled (pale gray color) to enabled (plain white color) is not noticeable enough and needs to be emphasized more. It could also be that out of all the possible buttons available the user considers the button closest to the last place that was interacted with, as they might expect that the controls to be used are grouped. The five bottom buttons might seem like they are part of another functionality as they are all relatively close to each other. As noted in problem 5 all the users finally found their way to the correct button after a while.

(P7) Users were confused when skipping a song resulted in playing the same song again. Some of the participants became confused when asked to skip a song. All correctly clicked the “skip song” button next to the “play/pause” button, but many clicked it more than once. Some users noted in the interview after the test that they were not able to hear that the song was indeed skipped when the “skip song” button was pressed. In about half of the tests the test conductor had to ask the participant to continue with the task, as they had, without knowing it, completed the step of skipping a song, and were taking too much time trying to figure out how to do what they had already done.

In all of the cases the program did skip the currently playing song but chose the next song to be the currently playing song resulting in the song starting from the beginning instead of playing a different song. This happened because of the low number of available songs of low intensity in the playlist used during all of the tests creating a high chance of playing the same song again when clicking the “skip song” button. Because of this a large part of the participants got the same song after correctly clicking “skip”, and thus became confused. This problem may be due to the playlist used in the test more than the usability of the program being at fault as all pressed the correct button on the first try within 3-5 seconds.

(P8) Users were unsure how to close the application. When asked to close the application, users were often hesitant to press the red cross to close the application. Users would often move the mouse near the cross, but would then ask if they should close the application using the red cross. We consider this problem to be a result of the test and not a real problem. We believe that by asking the users to do this relatively easy task last, they might have thought that we expected them to use something other than the red cross. They expected the task to be more difficult than it really were resulting in uncertainty and confusion.

14.4.2. Classification of problems

The severity of these problems have been classified as either cosmetic, serious or critical in table 14.1a. The classifications as well as their degree of severity is defined by Jesper Kjeldskov and Stage [11]. A cosmetic error is one that has a low impact on the use of the program. A serious error has a large impact on the use of the program and often causes a significant delay. A critical error is an error that completely stops a user from using the program. In addition to these classifications, we felt that it was necessary to include a “test error” classification to include problems that were caused mainly by the environment of the usability test itself.

Problem	Category
P1	Serious
P2	Cosmetic
P3	Serious
P4	Serious
P5	Serious
P6	Serious
P7	Cosmetic
P8	Test error

(a) Classification of the different usability problems

Problem class	Count
Critical	0
Serious	5
Cosmetic	2
Test error	1

(b) Numbers of errors

Table 14.1.: Problems classification and counts

14.4.3. Conclusion

As table 14.1a shows, the usability test uncovered several serious and a couple of cosmetic problems with the usability of the prototype. The serious problems would need to be addressed if development of the program were to continue. Solving some of the problems would not require many resources (e.g. P3 and P4) while resolving other problems might end up with a rather large portion of the user interface being altered and restructured (e.g. P5). Fixing the simple problems might include simple changes to the style and colors of certain controls while fixing the more complex problems might require functionality to be split up between several new screens. No critical problems were found during the test as all users completed both tasks within an acceptable timeframe and without receiving help from the test assistant or conductor. This indicates that the prototype is in a relatively useable state, that the interaction between program and user cannot be improved by much, and that doing so would not require many resources. All the users that were met with an error-window were also very quickly able to correct what they had done wrong by reading the error-message.

This test cannot conclude on any technical aspects of the prototype as only the interaction between the software and the user as well as the general usability of the program was tested. This conclusion is also solely based on the 8 described problems above as they were the most experienced and serious problems. Of some other minor problems that were noticed during some of the test were that one user thought that double-clicking an exercise plan in the list would trigger an action. This specific case, as well as other cases, was not thoroughly analyzed as they were only experienced one of the participants or seemed to be related to how the testing was carried out.

Fixing all of the found problems with the usability of the system would not take more than half a week for a couple of persons, but as the test was delayed, there would not be any time to reevaluate the changes with a second test. Further work on the prototype is therefore not recommended. But if the problems were to be addressed they would likely be dealt with as presented in the individual descriptions of the identified problems (see subsection 14.4.1).

15. Unit test

Another method used for testing is Unit testing, which takes single isolated methods and tests them by inputting some input for which a specific result is known and sees if the result produced by the method corresponds to the expected result. Sometimes methods are not entirely independent of other methods and isolating them for a Unit test requires the dependencies to be modeled with mock objects/methods. Mock objects/methods are stand-in objects/methods created to mimic that of the objects/methods used in the actual code. The mock objects/methods are designed to produce controlled results, which means that errors from the mock objects/methods that the method undergoing test depends on can be ruled out; hence isolating it from the rest of the code. Unit testing have been performed on key methods in our project, and the unit test cases performed, with explanations for each unit test, can be found in appendix C. As some tests produced results different from what was expected it made us aware of errors. All errors were corrected until all of the Unit tests produced results according to expectations.

16. Conclusion

The benefits of music are well known (see section 2.2), but are often not utilized to their full potential. Our application presents a novel way to use music both as a motivational tool and as a way to help the user execute an exercise plan. It does this by letting the user plan exercises and then playing music matching the intensity of the exercises when the user starts the plan (see section 2.5).

The prototype developed as part of this project was used to explore the usability of such a system. As the prototype only runs on Windows, these tests were mainly focused on the "planning" part of the application. Currently there are no tests on whether tempo is a understandable means of communication (i.e. whether users can reliably hear whether a song is faster or slower in tempo signaling him/her to run slower/faster).

Our tests showed a number of flaws in our application that should be amended if an application for mobile devices is to be developed. There were no critical usability errors, so no major changes to the interface design are needed.

Additional tests will be needed to be able to determine whether the application has an positive effect during exercise. We therefore recommend that additional tests are made where the user is exercising while using the application.

The application solves two main issues: it can unobtrusively guide you through an exercise plan and it can use music as a motivational tool. If the aforementioned test does not find any significant problems, we recommend building a mobile application based on the prototype.

17. Reflection

Several aspects of the developed program of this project can be improved. The prototype program, which currently only runs in a Windows environment should be developed for both Android and iOS so it is usable on the most popular mobile devices. In addition, we have several suggestions for improvements based on the usability test. We found five serious and two cosmetic usability problems which should be amended as described in subsection 14.4.3. Expanding the scope of the application to allow the user to keep track of playlists and songs on his/her device, as briefly touched upon in the analysis, would also be a possible way of improving the program.

An interesting further development to the system is to implement support for more advanced exercise equipment, such as heart rate monitor, GPS, temperature and other sensor systems to help guide the exerciser and control the exercise. These advanced pieces of equipment can possibly improve both the quality of the exercise and the possibilities for the exerciser.

Part IV.

Academic report

18. Process

This chapter will describe the process of the project and present possible improvements to be made for future projects.

18.1. Choice of method

Throughout the project period, planning and scheduling is essential for the final result. This chapter will describe what working method was used in this project. The choice of method is based on the received lessons in System Development of this semester and the textbook for that course[17].

Construction method

The construction method is based on rationality; analysis, decisions and action. It is a very general problem solving method because the process is very linear, and is therefore also often referred to as the waterfall model . The waterfall model, illustrated in figure 18.1, describes the construction method by the five top-down phases:

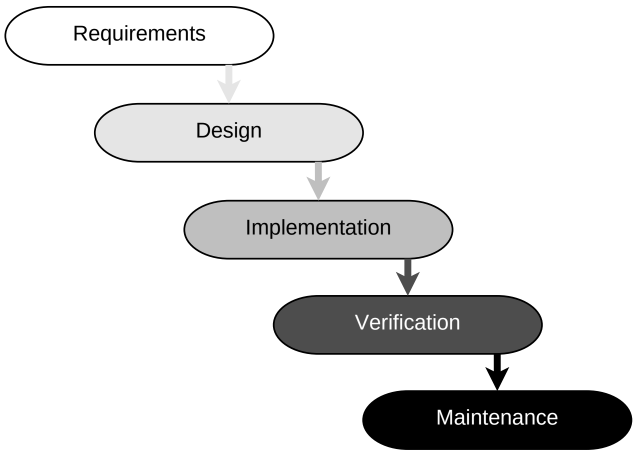


Figure 18.1.: The waterfall model

Requirements Describe the specification of requirements to the system

Design Describe the design specifications

Implementation Code the system

Verification Test the system

Maintenance Operating status of the system

This project is developed based partly on the construction method because of the benefits it brings to planning, scheduling and efficiency. Due to the relatively short amount of time available for the project, the construction method allowed for a more in-depth analysis of the different parts of the design phase, which resulted in less time wasted on switching between the phases. It is easier to schedule the entire project, since it is easier to anticipate how much time will be spent on the various phases. It is important to notice that a strictly followed waterfall method does not make room for moving up and down in the steps, and that the method followed in this project therefore only loosely follows the waterfall method. This all together allow for a more efficient project period, and allows for more work to be done in the same amount of time. In this semester we do not have the time and resources to maintain the system, so after completing the fourth step (Verification) the project was brought to an end.

18.2. Project scheduling

For project scheduling we initially scheduled the entire project period and wanted to take an iterative approach. Because we had trouble finding stakeholders we were quickly pushed behind schedule and had to pick an approach more similar to the waterfall model. We revised the schedule after this, but we did not make it very detailed. The new plan only showed when the general parts had to be done, and the smaller parts were done as we needed them. The various tasks were delegated on a day to day basis to the members willing to complete the task.

18.3. Stakeholders

Initially we contacted a number of running and exercising associations to meet users within our target group, however this was unsuccessful as most of these associations were not interested in helping us with our project, possibly because we asked too much of them. Our request was at first that they would use about 30 minutes on an interview and maybe participate in our usability test when the application was test-ready. Some time passed and we had only a few stakeholders to interview. We deemed that due to the circumstances and lack of time, this would have to do and used these interviews as a foundation for our requirements to the project.

18.4. Project usability testing

To test the usability of our application we did, as previously stated, try to contact some running and exercising associations to gain users who would participate in the usability testing of our application. This was unsuccessful however and as such we had to conduct our test using other methods. We instead contacted some shopping malls and heard if we could come to their mall with a small stand to ask their costumers to participate in our usability test. This of course was not the ideal situation, as we would have wanted the users to come to the university and use the usability lab located there. However, this might be a little too much to ask the users and the mall solution was more balanced as to how much effort the user had to put in to participate in our usability test. Conclusively we got an agreement set up with Friis Aalborg to come to their mall with the stand and conducted the test, in where we tested 10 potential users.

18.5. Communication in the group

Group meetings were held 3-5 times a week. On these meetings the tasks finished since last meeting were presented, new ideas were discussed and tasks for next meeting were delegated. We would also work on the program, testing it and building upon it, at these meetings. Our group worked on a trust system. There was no punishment for not doing a task, and it was expected that you would just do it. If someone repeatedly violated this trust, the group would conduct a meeting to try to figure out why the tasks were not completed. This system worked quite well, and it was not necessary to take any further action towards any group member. We did not have a designated leader role in the group, but some member would usually make the final decision on the task delegation when the group could not reach an unanimous agreement.

18.6. Communication with supervisor

Meetings with our supervisor were held about once a week while occasionally skipping a week. The last month of the project the supervisor was out of the country and no meetings with the supervisor in person were held in this period. In this period communication with the supervisor was achieved through e-mail. The day before a group meeting a draft report would be e-mailed to our supervisor together with a list of changes from last draft, and he would then read through it and comment on the changes at the meeting the following day. The supervisor would also provide us with resources for questions we might have. The supervisor was also available at e-mail for questions, and would always respond within a day or two.

18.7. The process improvement

For our next project it would be a good idea to have a more thorough time schedule. Even though it worked quite well with just an vague and generic schedule, there were times where we were not quite sure what needed to be done at that point in time. It would also be a good idea to get the stakeholders involved earlier. We had some trouble finding stakeholders and were mostly done with the design phase by the time we found some. As a result, they did not have much influence on the overall structure of the system. The same goes for users to test our application, we should try to get in contact with such users earlier in the process. The solution with a stand in a shopping mall have proved to be really effective and we might use this again for future projects.

19. K-means

As described earlier in chapter 11, we decided to use the k-means algorithm to cluster songs with similar tempo. The k-means algorithm was first described by MacQueen [15] and is an iterative refinement technique, meaning that it gradually approaches the final result as it runs through more iterations.

The k-means algorithm works by assigning each data point to a cluster, and then assigning a centroid to each cluster based on those points. The centroids can be seen as the average positions of the data points in a cluster. The algorithm then iteratively either updates the centroids of the clusters or the clusters of the points. This is done until an iteration is reached that yielded no change in centroids. This can be seen in algorithm 19.1.

Algorithm 19.1 General k-means algorithm[15]

```
1: procedure K-MEANS($P$, $k$)    ▷ The $k$ clusters of a set of points $P = \{p_1, p_2, \ldots, p_n\}$
2:     K-MEANS-INIT($P$, $k$)    ▷ Initialize cluster centroids $C = \{c_1, c_2, \ldots, c_k\}$
3:     while clusters are changing do
4:         for $i = 1, \ldots, n$ do    ▷ Update point clusters
5:             Assign point $p_i$ to the cluster $c_j$ whose center is closest
6:         for $j = 1, \ldots, k$ do    ▷ Update cluster centroids
7:             $n_j \leftarrow$ number of elements in $c_j$
8:             $c_j \leftarrow \frac{1}{n_j} \sum p_i \in c_j$
9:     return $C$
```

The algorithm leaves out the detail of how to initialize the clusters, though the final result is highly dependent on the initial clusters[10]. In the original MacQueen [15] paper, the first k initial points are used as initial cluster centroids. Because the k-means algorithm is so highly dependent on the initial clusters, much research have gone into uncovering better initialization methods[10].

It is generally best to choose initial centroids that are as close as possible to the resulting centroids[10]. This gives an advantage when working with 1-dimensional data as it, unlike multidimensional data, can be sorted. We initialize the clusters by first sorting the values from low to high. We then divide the values into k groups with an equal amount of values, and select the median of each group as the center for a cluster. This method can be seen in algorithm 19.2.

Algorithm 19.2 Initializing k-means

```
1: procedure K-MEANS-INIT($P$, $k$)
2:     SORT($P$)
3:     for $j = 1, \ldots, k$ do
4:         $s \leftarrow \frac{n}{k}(j + 0.5)$
5:         $c_j \leftarrow p_s$
6:     return $C$
```

Assuming uniform distribution, this should place each centroid reasonably close to the resulting position. If it is not a uniform distribution, the k-means algorithm will move the centroids to their resulting position through more iterations. This means that we use fewer iterations in the case where there is a uniform distribution.

Another area that the algorithm does not specify is how to determine what cluster is closest to a given point. Usually this is implemented as a simple linear search, as the number of clusters k is usually small. Though for large numbers of k, it is possible to achieve better performance by utilizing relevant data structures[10].

19.1. Runtime complexity of K-Means

Our application needs to be able to run with a small number of songs as well as a large number of songs. In order to evaluate how well the k-means algorithm performs as more songs are added, we can analyze the *asymptotic complexity* of the algorithm. In the following section, we calculate the worst case complexity of the algorithm. In this section, the notation k-means(n, k) will mean an invocation of the k-means algorithm with n elements and k clusters.

Our k-means implementation uses a sorting operation to initialize the clusters (as described in algorithm 19.2). There are numerous sorting algorithms, with usual implementations such as mergesort having a runtime complexity of $\Theta(n \log n)$[7]. Firstly the elements are sorted using mergesort, then the elements are divided into clusters, of which there are k, and each is initialized. The `kmeans-init` procedure is thus $\Theta(k + n \log n)$.

We then enter the main loop of the algorithm on line 3 of algorithm 19.1. This loop iterates as long as the centroids are changing. The amount of iterations necessary is highly dependent on the distribution and size of the input. We will refer to the number of iterations necessary as I.

On line 4 of algorithm 19.1, we enter the point update loop. Here we find the closest cluster for all n points. As explained in chapter 11, this is done with a linear search through the clusters. Thus the complexity of this loop is $\Theta(nk)$.

On line 6 of algorithm 19.1, we enter the cluster update loop. Here we find the centroid of all the points assigned to each cluster. We assume that the run time for acquiring the number of elements in c_j is constant. On line 8 we do a summation of all the points assigned to c_j. As all points are assigned to exactly one point, this will sum all points exactly once. Thus the complexity of the loop on line 6 of algorithm 19.1 is $\Theta(n + k)$.

As the loop from line 4 and 6 are done I times, we can calculate the total complexity like so:

$$\text{k-means}(n, k) = \Theta(k + n \log n + I(nk + n + k))$$

We can simplify the above using the fact that $k \leq n$, since there must be at least k elements for there to be k clusters.

$$\text{k-means}(n, k) = \Theta(n \log n + Ink)$$

19.2. Estimating $\Theta(I)$

As the number of iterations required depends greatly on the input, knowing the asymptotic complexity of I (and thus also the entire algorithm) is very difficult from a theoretical standpoint.

What we can do instead, is to empirically estimate the number of iterations by running the algorithm on a computer and counting the number of iterations. To do this, it is necessary to make assumptions on what the input will be. Our application will only use 1-dimensional data and will only divide the data into three clusters. This is therefore not a test of k-means, but will instead be referred to as 3-means.

In addition, we assume the values have a uniform distribution between zero and one. While this may or may not be representative of the actual distribution of tempo in a playlist, it is amongst the worst case for the input of k-means as it does not have any clusters. The algorithm would thus be expected to perform poorer than a distribution that contains clusters.

Our test also includes the initialization method in algorithm 19.2. Due to this initialization method and the assumptions listed in the paragraph above, the results are only applicable for our specific application.

The test was run with doubling number of elements starting at 4 (e.g. the 1st was with 4 elements, the 2nd was with 8 elements, etc.). For each number of elements, 1000 tests with different random values was run and analyzed to find the mean. These results, along with the standard deviation, can be seen in table 19.1.

Elements	Iterations		±%
4	1.098	±0.297	27.1
8	1.256	±0.557	44.4
16	1.436	±0.862	60.0
32	1.815	±1.147	63.2
64	2.357	±1.638	69.5
128	2.982	±1.900	63.6
256	3.597	±2.310	64.2
512	4.503	±2.791	62.0
1024	5.423	±3.291	60.7
2048	6.253	±3.605	57.7
4096	7.252	±4.018	55.4
8192	8.463	±4.311	50.9

Table 19.1.: Results of test of 3-means

In figure 19.1, we have graphed these results in a double logarithmic graph. We can see that the mean values follow a straight line, indicating that there is a relation of the form $y = ax^b$ for some constant b between the two axes. This indicates that for this specific average case, it is likely that $I(n) = \Theta(n^b)$. From table 19.1, we can see that at 1000 elements it only took 5±3 iterations, which can be done almost instantaneously on most modern computers and mobile devices. We do not expect more than about 1000 songs to be on a single mobile device at a time, so this algorithm has a suitable performance for our application.

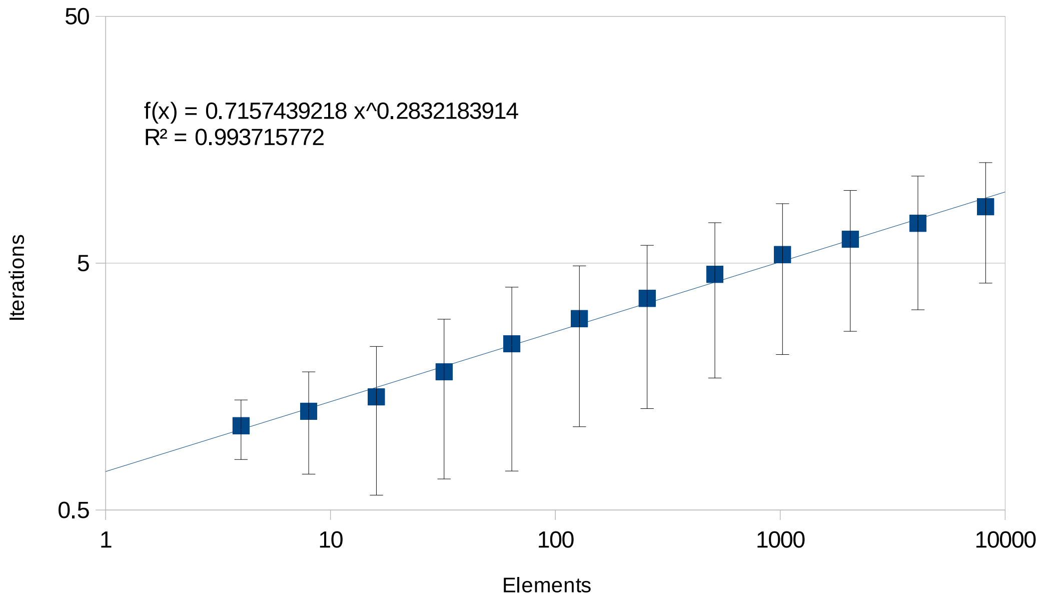


Figure 19.1.: Number of elements vs. number of iterations required for 3-means. The error bars represent the standard deviation from the mean.

Bibliography

[1] *Android design standards. URL* `http://developer.android.com/design/index.html`.

[2] *The echonest. URL* `http://the.echonest.com/`.

[3] *Task parallelism (task parallel library), . URL* `http://msdn.microsoft.com/en-us/library/dd537609.aspx`.

[4] *Introduction to wpf, . URL* `http://msdn.microsoft.com/en-us/library/aa970268.aspx`.

[5] *Aagaard. About aagaard. URL* `http://www.marinaaagaard.dk/idd41.asp`.

[6] *David Benyon.* Designing Interactive Systems. *Pearson, 2 edition, 2010.*

[7] *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.* Introduction to algorithms. *The MIT press, 3 edition, 2009.*

[8] *Jun Gong and Peter Tarasewich. Guidelines for handheld mobile device interface design. In* In Proceedings of the 2004 DSI Annual Meeting*, 2004. URL* `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.87.5230`.

[9] *Masataka Goto and Yoichi Muraoka. Music understanding at the beat level - real-time beat tracking for audio signals.* IJCAI-95 Workshop on Computational Auditory Scene Analysis*, 1997.*

[10] *Greg Hamerly and Charles Elkan. Alternatives to the k-means algorithm that find better clusterings. In* Proceedings of the eleventh international conference on Information and knowledge management*, 2002. URL* `http://charlotte.ucsd.edu/users/elkan/cikm02.pdf`.

[11] *Mikael B. Skov Jesper Kjeldskov and Jan Stage, editors.* DOES TIME HEAL? A LONGITUDINAL STUDY OF USABILITY*, November 2005. Aalborg University, Department of Computer Science.*

[12] *C. I. Karageorghis, P. C. Terry, and A. M. Lane. Development and initial validation of an instrument to assess the motivational qualities of music in exercise and sport: the brunel music rating inventory.* J Sports Sci*, 17(9):713–724, Sep 1999. doi: 10.1080/026404199365579. URL* `http://dx.doi.org/10.1080/026404199365579`.

[13] *Costas I. Karageorghis and David-Lee Priest. Music in the exercise domain: a review and synthesis (part i).* International Review of Sport and Exercise Psychology*, 5(1):44–66, Mar 2012. doi: 10.1080/1750984X.2011.631026. URL* `http://dx.doi.org/10.1080/1750984X.2011.631026`.

[14] *Costas I. Karageorghis and David-Lee Priest. Music in the exercise domain: a review and synthesis (part ii).* International Review of Sport and Exercise Psychology, *5(1):67–84, Mar 2012. doi: 10.1080/1750984X.2011.631027. URL* `http://dx.doi.org/10.1080/1750984X.2011.631027`.

[15] *J. MacQueen. Some methods for classification and analysis of multivariate observations. In* Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, *volume 1, pages 281–297. University of California Press, 1967. URL* `http://projecteuclid.org/euclid.bsmsp/1200512992`.

[16] *Tom MacWright. Literate jenks natural breaks and how the idea of code is lost, 02 2013. URL* `http://macwright.org/2013/02/18/literate-jenks.html`.

[17] *Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage.* Object-oriented Analysis & Design. *Marko Publishing, 2000. ISBN 8777511506.*

[18] *Microsoft. Sql server compact edition. URL* `http://www.microsoft.com/en-us/sqlserver/editions/2012-editions/compact.aspx`.

[19] *Pacing Technologies. Pacedj faq. URL* `http://www.pacedj.com/faq/`.

[20] *Wikipedia user Rezonansowy. Boundless informant data collection, 8 2013. URL* `http://en.wikipedia.org/wiki/File:Boundless_Informant_data_collection.svg`.

Part V.

Appendix

A. Preplanned exercise plans

Easy exercise program (for the unskilled exerciser)
15 min. warm up (low intensity)
1 minute (interval) - relax/recovery pace (medium intensity)
2 minute (interval) - high pace exercise (moderate/high- intensity)
4 - 5 intervals repetitions
10 min. cool-down (low intensity)
Exercise time in total: 37 - 40 minute
Medium exercise program (for the mid-skilled exerciser)
15 min. warm up (low intensity)
0,5 minute (interval) - relax/recovery pace (medium intensity)
1 minute (interval) - high pace exercise (moderate/high- intensity)
5 - 7 intervals repetitions
10 min. cool-down (low intensity)
Exercise time in total: 32,5 - 35,5 minute
Hard exercise program (for the skilled exerciser)
15 min. warm up (low intensity)
1,5 minute (interval) - relax/recovery pace (medium intensity)
0,5 minute (interval) - high pace exercise (moderate/high- intensity)
10 intervals repetitions
10 min. cool-down (low intensity) low
Exercise time in total: 45 minute

B. Functional Test Plan

B.1. Test Items

The tested items in this project is the main program, this includes the GUI and the methods that this controls, both those concerning plans as well as music. Specifically this is the application specified by the requirements specification in section 2.4.

B.2. Features to be tested

In the test the GUI will be in focus, the following features will be tested:

- Create exercise plan
- Modify exercise plan
- Delete exercise Plan
- Play exercise plan
- Skip song
- Pause song
- Stop exercise plan
- Exit application (save changes)

B.3. Features not to be tested

As focus is mostly on the GUI

- Timing in changing songs
  - The timing between changing songs compared to the time set in the exercise plan.

B.4. Approach

To test the GUI as much as possible every task should be done for every button leading to the tested feature, e.g. the create exercise plan test should be conducted using both the button on the main menu and also the button on the plan picker screen. The different tests should also include the use of all ways to do the task, e.g. use both the sliders and the textboxes to set the durations for an exercise plan, as both ways are viable in the application and both should yield the same result.

Each test should be completed with a number of different values for which the result is known and then confirm that the actual result is as expected.

B.5. Item pass/fail criteria

For a test to pass the task concerning each feature/item should complete without errors, exceptions or similar. The test should also yield the expected result no matter how you perform the task.

For a test to fail, the test should result in a error or exception to occur or the result should be different from the expected.

B.6. Testing tasks

Create exercise plan

1. Go to exercise plan screen
   - Use the “Create exercise plan”
   - Click the “Start exercise plan” and then click the “add” button
2. Input name, durations and interval repetitions (durations can be input using either the sliders or the textboxes)
3. Save the exercise plan
4. Verify that the exercise plan is created with the input values by closing the application and opening the exercise plan database in a database editor.

Modify exercise plan

1. Go to exercise picker screen
2. Select an existing plan and push the “Modify” button
3. Change the values of the exercise plan
   - Name should be changed
   - Duration should be changed
   - Number of intervals should be changed
4. Save the exercise plan
5. Verify that the exercise plan is created with the input values by closing the application and opening the exercise plan database in a database editor

Delete exercise Plan

1. Go to exercise picker screen
2. Select an existing plan and push the “Delete” button
3. Verify that the exercise plan is deleted by closing the application and opening the exercise plan database in a database editor

Play exercise plan

1. Go to exercise picker screen
2. Select an existing plan and push the “Play” button
3. Verify that the playlist is playing according to the information stored in the exercise plan by opening the exercise plan database in a database editor

Skip song

1. Go to exercise picker screen
2. Select an existing plan and push the “Play” button
3. Press the “Skip” button on the play screen
4. Verify that the current song is stopped and the next song plays

Pause song

1. Go to exercise picker screen
2. Select an existing plan and push the “Play” button
3. Press the “Play/Pause” button on the play screen
4. Verify that the song has stoped and wait for some time
5. Press the “pause” button and verify that the song continues from where it was paused.

Stop exercise plan

1. Go to exercise picker screen
2. Select an existing plan and push the “Play” button
3. Press the “Stop” button and verify that the song stops and you are returned to the exercise plan picker screen

Exit application (save changes)

1. Note the state of the current exercise plans
2. Make some visible changes to the exercise plans database

    - Make a new plan
    - Delete a plan
    - Modify a plan

3. Exit the application
    - Press the “X”-mark in upper-right corner of the window
    - Return to the start screen and press the “Exit” button.

B.7. Environmental needs

The application prototype should be run from a PC running Windows 7/8, the PC should have the latest .NET framework installed.

C. Unit test cases

EchonestLimiterTest

The Echo Nest API implementation features a limiter for limiting the number of requests that the system can send to The Echo Nest service depending on the quota the used key have. We would like to assure that the limiter actually limits the number of requests in a way that makes "useless" calls, where the quota have been exceeded, happen as little as possible. The code have been tested by measuring the time it takes to make a number of actions while using the limiter and comparing this to the expected time used, this have been done using the code listed in listing 5.

Listing 5: EchonestLimiterTest

```
public void EchonestLimiterTest()
{
  Limiter TestLimiter = new Limiter(30);
  DateTime now = DateTime.Now;
  for (int i = 0; i < 10; i++)
  {
    TestLimiter.Delay().Wait();
  }
  Assert.IsTrue(DateTime.Now >= now +
  TimeSpan.FromSeconds(60 / (30 / 10)));
}
```

EchonestCacheCallsTest

The Echo Nest service is cached to limit the calls to the service. The caching is handled by the EchonestCache, which implements the IEchonestService, which means it can be used just as a The Echo Nest API alone would. The EchonestCache does only need to request the same information once from The Echo Nest service. Therefore we need to ensure that the cache does only call The Echo Nest service once. For this test The Echonest API service have been *stubed*, which means that the API service have been replaced with code that have a known return value. The stub is as seen in listing 6.

Listing 6: EchonestStub, implements IEchonestService and returns a known TrackSummary

```
public class EchonestStub : IEchonestService
```

```
{
  public int calls = 0;
  public async Task<TrackSummary>
  GetSummary(string artist, string title)
  {
    calls++;
    return new TrackSummary()
    {
      Tempo = 10;
      CoverURL = "Testytesto";
      Energy = 0.5;
    };
  }
}
```

The stub returns a known TrackSummary, while also implementing a new local variable, calls, which contains the number of calls to the GetSummary method. This is needed to check that the EchonestCache does only contact the IEchonestService once and therefore have cached the information recieved from the first call. This is tested with the code shown in listing 7.

Listing 7: EchonestCacheCallsTest, tests that the EchonestCache only calls the IEchonestService once

```
public void EchonestCacheCallsTest()
{
  if (File.Exists("TestDatabase.sdf"))
  {
    File.Delete("TestDatabase.sdf");
  }
  EchonestStub stub = new EchonestStub();
  EchonestCache cache = new EchonestCache("DatabaseConnection",
     stub);
  TrackSummary summary = cache.GetSummary("Test and the Testons",
     "Test me one more time").Result;
  summary = cache.GetSummary("Test and the Testons", "Test me one
     more time").Result;
  Assert.IsTrue(stub.calls == 1);
  cache.Dispose();
}
```

EchonestCacheCorrectnessTest

Another thing that needs to be ensured about the EchonestCache is the correctness of the cache. The information requested from The Echo Nest Service, and that from the EchonestCache should be the same. To test this the stub in listing 6 is used and the values requested from the EchonestCache is compared to the values known to be stored in the stub, this is done in the program code seen in listing 8.

Listing 8: EchonestCacheCorrectnessTest, tests the correctness of the EchonestCache

```
public void EchonestCacheCorrectnessTest()
{
  if (File.Exists("TestDatabase2.sdf"))
  {
    File.Delete("TestDatabase2.sdf");
  }
    TrackSummary template = new TrackSummary()
  {
Tempo = 10, CoverURL = "Testytesto", Energy = 0.5 };
  EchonestStub stub = new EchonestStub();
  EchonestCache cache = new EchonestCache("DatabaseConnection2",
      stub);
  TrackSummary summary = cache.GetSummary("Test and the Testons",
      "Test me one more time").Result;
  Assert.IsTrue(summary.CoverURL == template.CoverURL &&
      summary.Energy == template.Energy && summary.Tempo ==
      template.Tempo);
  cache.Dispose();
}
```

ExercisePlanDurationTest

The ExercisePlan contains information about the user made plan, how long warm-up, cool-down and intervals should be as well as how many repetitions the intervals should have. The total amount of time for a ExercisePlan can be calculated using all these valu

Listing 9: PrototypeKMeansClusterTest

```
public void PrototypeKMeansClusterTest()
{
  Random rand = new Random();
  List<KeyValuePair<double, string>> values = new
      List<KeyValuePair<double, string>>();
  for (int i = 0; i < 10; i++)
  {
    values.Add(new KeyValuePair<double,
        string>(rand.NextDouble(), "testlow"));
    values.Add(new KeyValuePair<double, string>(rand.NextDouble()
       + 2.5, "testmedium"));
    values.Add(new KeyValuePair<double, string>(rand.NextDouble()
       + 4, "testhigh"));
  }
  var result = Clustering.Cluster(50, 3, values.ToArray(), x =>
      x.Key, x => x.Value).OrderBy(x => x.Key);

  Assert.AreEqual(0.5, result.ElementAt(0).Key, 0.1);
  Assert.AreEqual(3, result.ElementAt(1).Key, 0.1);
  Assert.AreEqual(4.5, result.ElementAt(2).Key, 0.1);
}
```

es of time by using the ExercisePlan.Duration method. This should be equal to the actual time of the entire plan, this can be seen tested in listing 10

Listing 10: ExercisePlanDurationTest1, tests that the duration of a plan is correctly calculated

```
public void ExercisePlanDurationTest1()
{
  ExercisePlan plan = new ExercisePlan("testplan")
  {
    CoolDownDuration = TimeSpan.FromSeconds(250.3),
    HighIntervalDuration = TimeSpan.FromSeconds(1000.23),
    LowIntervalDuration = TimeSpan.FromSeconds(0.0),
    Repetitions = 4,
    WarmUpDuration = TimeSpan.FromSeconds(3)
  };
  Assert.AreEqual(4254.22, plan.Duration.TotalSeconds, 0.5);
}
```

The test at first showed that the duration was faulty calculated, it was discovered that this was due to faulty code, where a variable had been used where another should have been. This was corrected resulting in correct calculation of the plan duration.

ExercisePlanIntensityIntervalTest

The ExercisePlan contains a method for creating a List of KeyValuePairs with the end time of a intensity level as key and the intensity as the value. This is used to keep track of the plan while executing it and as of such is of crucial importance to the system. The plan is generated from the values for warm-up, cool-down as well as intervals in the ExercisePlan. To test that this method returns the correct values, we manually create a List that matches the ExercisePlan and compares it to the result returned from the IntensityIntervals method, this is done as seen in listing 11.

Listing 11: ExercisePlanIntesityIntervalTest

```
public void ExercisePlanIntensityIntervalTest()
{
  ExercisePlan plan = new ExercisePlan("testplan")
  {
    WarmUpDuration = TimeSpan.FromSeconds(1),
    HighIntervalDuration = TimeSpan.FromSeconds(15),
    LowIntervalDuration = TimeSpan.FromSeconds(30),
    Repetitions = 10,
    CoolDownDuration = TimeSpan.FromSeconds(150)
  };
  List<KeyValuePair<TimeSpan, Intensity>> expectedKeyValuePair =
     new  List<KeyValuePair<TimeSpan,Intensity>>();
  expectedKeyValuePair.Add(new KeyValuePair<TimeSpan,
     Intensity>(TimeSpan.FromSeconds(1), Intensity.Low));
  for (int i = 0; i < 10; i++)
  {
    expectedKeyValuePair.Add(new KeyValuePair<TimeSpan,
       Intensity>(TimeSpan.FromSeconds(31+i*45),
       Intensity.Medium));
    expectedKeyValuePair.Add(new KeyValuePair<TimeSpan,
       Intensity>(TimeSpan.FromSeconds(46+i*45), Intensity.High));
  }
  expectedKeyValuePair.Add(new KeyValuePair<TimeSpan,
     Intensity>(TimeSpan.FromSeconds(601), Intensity.Low));
  IEnumerable<KeyValuePair<TimeSpan, Intensity>> actual =
     plan.IntensityIntervals();

  CollectionAssert.AreEqual(expectedKeyValuePair,
     actual.ToList());
}
```

PrototypeKMeansClusterTest

The KMeans.Cluster method, described in chapter 19, calculates three centroids which each is the average of one of three clusters calculated from the intensity of the songs given as a parameter for the method. In our system this is used to calculate which intensity group each song should belong to. This method is tested by running the algorithm on points generated in three specific intervals which should produce a centroid approximately in the middle of each interval. The code for testing this is shown in listing 12.

Listing 12: PrototypeKMeansClusterTest

```
public void PrototypeKMeansClusterTest()
{
  Random rand = new Random();
  List<KeyValuePair<double, string>> values = new
     List<KeyValuePair<double, string>>();
  for (int i = 0; i < 10; i++)
  {
    values.Add(new KeyValuePair<double,
       string>(rand.NextDouble(), "testlow"));
    values.Add(new KeyValuePair<double, string>(rand.NextDouble()
       + 2.5, "testmedium"));
    values.Add(new KeyValuePair<double, string>(rand.NextDouble()
       + 4, "testhigh"));
  }
  var result = Clustering.Cluster(50, 3, values.ToArray(), x =>
      x.Key, x => x.Value).OrderBy(x => x.Key);

  Assert.AreEqual(0.5, result.ElementAt(0).Key, 0.1);
  Assert.AreEqual(3, result.ElementAt(1).Key, 0.1);
  Assert.AreEqual(4.5, result.ElementAt(2).Key, 0.1);
}
```

D. Usability test notes

This is the summary of the notes taken during the usability tests, video documentation is attached on the following CD.

Test user #1: Male

IT Experience: daily user

No video documentation.

Used sliders to chose durations of each part exclusively.

Comment - Problem with mouse sensitivity or sliders being too sensitive/hard to control duration.

Test user #2: Female

IT Experience: daily user

Bad/no video documentation (20131211_1239.avi).

Task 1 - Unable to locate "create new exercise plan", went through the program at first, was confused of what to do. When she found out what to do she quickly did the task, she used only the textboxes to input the duration of each part of the exercise.

Task 2 - No problems in starting the plan.

Comment - She would like some more information on how to use the program, she was very confused at first.

Test user #3: Male

IT Experience: Average

Video documentation (20131211_1245.avi).

Task 1 - Fast start, used sliders, but had precision problems due to mouse/slider sensitivity.

Task 2 - Detour through "add new plan" at "start exercise" screen.

Comment - Confusing labels on the "plan maker" screen (in Danish version of the program), very good program besides that.

Test user #4: Female

IT Experience: Very high

video documentation (20131211_1253.avi).

Task 1 - Overlooks name-textbox and gets a error message, hereafter completes the task (used her own name as plan name). Uses sliders, is very fast (used the touchpad)

Task 2 - Confusing on how to get to play screen.

Comment - Unclear how to get to play screen.

Test user #5: Male

IT Experience: Average

video documentation (20131211_1302.avi).

Task 1 - Does not understand the task at first. Uses sliders, first was slow but gets faster. Overlooks name textbox

Task 2 - Confusion in "plan selection", double-clicks on the desired plan to start, takes detour into "add plan screen". Finds start button afterwards.

Comment - High sensitivity slider/mouse, easy application.

Test user #6: Male

IT Experience: Average

video documentation (20131211_1311.avi).

Task 1 - Used own name as plan name. Starts by using sliders, but goes over to using textboxes to fill in all of the durations.

Task 2 - Fast completion.

Comment - none

Test user #7: Female

IT Experience: Average

video documentation (20131211_1320.avi).

Task 1 - Uses own name as plan name. Hovers over slider, but sees the textbox and fills in the durations by using these.

Task 2 - Fast completion.

Comment - Missing exit button ("should I just use the X?").

Test user #8: Female

IT Experience: Basic

video documentation (20131211_1340.avi).

Task 1 - Uses sliders to input durations, hard to be precise, a little slow. Fills in name as the last thing.

Task 2 - Fast on play screen, tries to skip after pausing music.

Comment - Confused on how to close the program, think she has to go back in the application.

Test user #9: Female

IT Experience: Average

video documentation (20131211_1355.avi).

Task 1 - Uses own name as plan name. Uses slider, no problems by accuracy (touchpad).

Task 2 - Fast to play screen. Tries to click on song-title.

Comment - Could not hear music changed song.

Test user #10: Male

IT Experience: Very high

video documentation (20131211_1421.avi).

Task 1 - Starts with sliders and changes to textbox

Task 2 - Fast completion

Comment - Fewer buttons

E. Usability test tasks

Task 1:

Create new exercise plan, with the following settings:

Name:	Plan1
Warm-up duration:	5:00
Resting interval duration:	0:20
Working interval duration:	0:40
Repetitions:	4
Cool-down duration:	5:00

Task 2:

Start exercise plan “Easy plan” and skip a song. Hereafter stop the exerciser plan and terminate the program.

F. Statement of consent (Danish)

Jeg indvilger i at deltage i denne undersøgelse og er indforstået med, at jeg må trække mig fra undersøgelsen på et vilkårligt tidspunkt, hvis jeg skulle ønske dette.

Jeg indvilger i at mit brug af programmet samt lyd ved undersøgelsen bliver optaget forudsat at denne optagelse kun bliver brugt i forsknings eller undervisningssammenhæng.

Jeg indvilger i at forskningsdata indsamlet til studiet må blive publiceret eller videregivet til andre forskere, på betingelse af at anonymiteten opretholdes.

Dato: 11 / 12 - 2013

Navn: ____________________________

Underskrift: ____________________________

Denne test er udarbejdet af datalogi gruppe, ds301e13 bestående af følgende medlemmer: Anders Hermansen, Joachim Klokkervoll, Lynge Poulsgaard, Mike Pedersen & Samuel Nygaard Pedersen

Statement of consent (translated to English)

I agree to participate in this test and understand that I can withdraw from the test at any time.

I agree that my use of the program and the sound of the test will be recorded assuming that the recording is used only for research or educational purposes.

I agree that research data collected in the test may be published or given to other researchers on condition of anonymity maintained.

Date: 11 / 12 - 2013

Name: ____________________________

Signature: ____________________________

This test is developed by computer science group ds301e13 consisting of: Anders Hermansen, Joachim Klokkervoll, Lynge Poulsgaard, Mike Pedersen & Samuel Nygaard Pedersen