



TEMPO PLAYER - THE MUSIC PLAYER THAT FITS YOUR TEMPO  
**IVAN:** METAPHOR - THINK ABOUT MORE SUITABLE TITLE.

P8 PROJECT  
GROUP SW802F15  
SOFTWARE  
DEPARTMENT OF COMPUTER SCIENCE  
AALBORG UNIVERSITY  
SPRING 2015





**AALBORG UNIVERSITY**  
STUDENT REPORT

**Department of Computer Science**

Selma Lagerløfs Vej 300

9220 Aalborg Ø

Phone (+45) 9940 9940

Fax (+45) 9940 9798

<http://www.cs.aau.dk>

**Title:**

Tempo Player - The music player  
that fits YOUR tempo

**Subject:**

Mobile Systems

**Project period:**

02-02-2015 – 27-05-2015

**Project group:**

SW802F15

**Participants:**

Alexander Drægert  
Christoffer Nduru  
Dan Petersen  
Kristian Thomsen

**Supervisor:**

Ivan Aaen

**Abstract:**

The purpose of this project was to make a runner's running experience less tedious and more inspiring, by playing music which fits their running tempo. The project is based on the research of Edworthy and Waring, which concluded that music tempo affects a runner's pace and enjoyment.

**Printings: ?**

**Pages: ??**

**Appendices: ?**

**Total pages: ??**

**Source code:**

<https://github.com/SW802F15/SourceCode/tree/???????>

*The content of the report is freely available, but may only be published (with source reference) with consent from the authors.*



# Preface

---

Aalborg, April 24, 2015

---

Alexander Drægert

---

Christoffer Nduru

---

Dan Petersen

---

Kristian M. Thomsen



# Contents

---

<b>Preface</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Methodology</b>	<b>3</b>
2.1 eXtreme Programming explained . . . . .	3
2.2 eXtreme Programming compared to SCRUM . . . . .	3
2.3 eXtreme Programming as applied in this project . . . . .	3
2.4 Modifications of eXtreme Programming . . . . .	3
<b>3 Tempo Player - The Tour</b>	<b>5</b>
<b>4 Music Player</b>	<b>7</b>
4.1 Use Cases . . . . .	7
4.2 Implementation . . . . .	7
4.3 Methodology Observations . . . . .	7
<b>5 Song Scanner</b>	<b>9</b>
5.1 Use Cases . . . . .	9
5.2 Implementation . . . . .	9
5.3 Methodology Observations . . . . .	9
<b>6 Step Counter</b>	<b>11</b>
6.1 Use cases . . . . .	11
6.2 Specification . . . . .	11
6.3 Implementation . . . . .	11
6.4 Methodology Observations . . . . .	12
<b>7 Test</b>	<b>13</b>
7.1 Acceptance Test . . . . .	13
7.2 Unit Test . . . . .	13
7.3 Test - Reflection . . . . .	13
<b>8 Methodology - Reflection</b>	<b>15</b>
8.1 Introduction . . . . .	15
8.2 40-hour Work Week . . . . .	15
8.3 Coding Standard . . . . .	15
8.4 Collective Ownership . . . . .	15
8.5 Continuous Integration . . . . .	15
8.6 Metaphor . . . . .	15
8.7 On-Site Costumer . . . . .	15
8.8 Pair Programming . . . . .	15

8.9	Planning . . . . .	15
8.10	Refactoring . . . . .	15
8.11	Simple Design . . . . .	15
8.12	Small Releases . . . . .	15
8.13	Testing . . . . .	15
8.14	Other . . . . .	15
<b>9</b>	<b>Conclusion</b>	<b>17</b>
9.1	Discussion . . . . .	17
9.2	Future Work . . . . .	17
9.3	Conclusion . . . . .	17
<b>10</b>	<b>Temporary Work Sheets</b>	<b>19</b>
10.1	Methodology . . . . .	19
10.2	Refactoring . . . . .	43
<b>A Releases - (After Methodology)</b>		
<b>B Tactile Interface - (After Step Counter)</b>		
<b>Bibliography</b>		
<b>C Project CD</b>		
<b>D Examples &amp; ToDo</b>		



# Introduction

# 1

Running is a popular form of exercise, however, it can be a tedious and uninspiring endeavour. To improve the experience, Edworthy and Waring [2] found that “... *participants enjoyed what they were doing [running] more when they were listening to music of any sort when compared to when they were not.*”

It was further concluded by Edworthy and Waring [2] that the volume and tempo of the music influenced the running experience. They concluded that the running pace for novice runners, while listening to relatively low-tempo music, was slower than when not listening to music. Additionally listening to high-tempo music resulted in a faster running pace, compared to when not listening to music.

This conclusion is in disagreement with the conclusion of Yamamoto et al. [4] which suggests, that “... *music had no impact on mean power output*”.

As a result, we can not definitively conclude whether music of different tempo will affect the running experience differently. However by adhering to Edworthy and Waring [2]’s conclusion, we can only improve the running experience, since Yamamoto et al. [4] concludes there can be no negative impact, by playing music.

Today many runners use their smartphone as a music player, which can either be placed in their hand, pocket, or on their arm. The sensors in a smartphone enable monitoring the pace and speed of the runner. From this knowledge the first problem can be stated:

*How can we provide music with an appropriate tempo, compared to the current pace, to the runner through the use of a smartphone?*

Operating a smartphone while running is difficult, especially if it is placed in the pocket or on the arm of the runner. In order for the runner to operate the smartphone properly, the runner would have to stop running, or focus more than normally, which can disrupt the runner’s form, which can lead to injuries and accidents.

From this knowledge the second problem can be stated:

*How can a smartphone application be operated without disrupting the runner’s form or concentration?*

**Alexander:** BPM error deviation = 7.5 BPM - <http://www.cs.virginia.edu/~stankovic/psfiles/MusicalHeart-SenSys2012-CameraReady.pdf>

**Ivan:** Write about XP

**Ivan:** Deliberate on what Yamamoto et al. [4] is about.

# Methodology 2

---

## 2.1 eXtreme Programming explained

What is the XP methodology? What does Extreme mean? E.g. Review is much more important and useful, so turn it up in comparison to other methodologies. Start of Beck books.

## 2.2 eXtreme Programming compared to SCRUM

What are the differences in philosophies? E.g. SCRUM is all management, whereas XP is both programming and management. E.g. Different philosophies about review, quality, etc.

## 2.3 eXtreme Programming as applied in this project

What are the principles of XP. What is the goal of XP. How do we expect it to be used?

## 2.4 Modifications of eXtreme Programming

Our modifications of XP. Practical applications.

e.g. Code Standards are not as important in our project because we are familiar with each other.

p.s. remember to mention how we have used **Essence** retrospective methods.

p.p.s. remember to mention that we use TDD, because it is referred to from Test - Unit Test.



# Tempo Player - The Tour 3

---

This chapter should describe an overview of the program, as well as provide a couple of screenshots detailing the GUI of the program.

i.e. Tempo Player is a music player that utilises the accelerometer to count the steps of the user. Tempo Player then compares this pace with the beats per minute of songs collected in a SQL Lite database, and plays the one most similar. The beats per minutes of each song are found by querying a web service.



# Music Player 4

---

## 4.1 Use Cases

This section should describe the use cases set up for the Music Player feature.

i.e. The Music Player should play the next song after completion of the current playing song.

## 4.2 Implementation

This section should describe the programmatically interesting and relevant parts of the Music Player.

i.e. We found that trying to stop the music player while not loaded caused it to skip to next song. It was due to the fact this was an illegal action in that state, as seen in Figure (Android graph over mediaplayer states). We fixed this issue by implementing something, as seen in Listing (Code snippet of interesting code).

## 4.3 Methodology Observations

This section should contain the observation about the XP methodology gathered during the development of the Music Player.

i.e. When developing the music player we have trouble adhering to the **Simple Design** principle, which states “Only develop what is need now”, because we know we will implement additional features later, which lead us to implement methods for future use.





# Song Scanner 5

---

## 5.1 Use Cases

This section should describe the use cases set up for the Music Player feature.

i.e. The Music Player should play the next song after completion of the current playing song.

## 5.2 Implementation

This section should describe the programmatically interesting and relevant parts of the Music Player.

i.e. We found that trying to stop the music player while not loaded caused it to skip to next song. It was due to the fact this was an illegal action in that state, as seen in Figure (Android graph over mediaplayer states). We fixed this issue by implementing something, as seen in Listing (Code snippet of interesting code).

## 5.3 Methodology Observations

This section should contain the observation about the XP methodology gathered during the development of the Music Player.

i.e. When developing the music player we have trouble adhering to the **Simple Design** principle, which states “Only develop what is need now”, because we know we will implement additional features later, which lead us to implement methods for future use.



# Step Counter 6

---

In the following sections, the development of our step counter module will be explained.

**Christoffer:** Intro omskrives?

## 6.1 Use cases

## 6.2 Specification

## 6.3 Implementation

In our implementation of the step counter we decided to split the task into a data collection task, testing of algorithm task and an implementation on android task. The reason for doing so is that our approach writing tests, code, and then run the test is unsuited for implementing a step counter. This is because the running of tests cannot be automated as it requires a person to move around with the smartphones.

As there is no built-in step counter library available for Android API version 16, we had to choose an algorithm. We found the precision could vary greatly from algorithm to algorithm, so we had to test algorithms against each other. In order to do so we first had to generate some sample data collected from the smartphones' sensors.

### 6.3.1 Data Collection

The first step of data collection is to figure out what data to collect, to do this we take a look at the available sensors on an Android smartphone, see Android [1] for an overview of these. We decided to gather data from three sensors: accelerometer, gravity and gyroscope. We also measured the time delay between each sensor reading.

#### 6.3.1.1 Collection Procedure

To generate the sample data we had a person perform the following tasks while carrying the smartphone in a specific position (in hand, in pocket or strapped to arm)

1. Walk 100 steps. *Walk: to advance on foot at a moderate speed; proceed by steps; move by advancing the feet alternately so that there is always one foot on the ground.*
2. Jog 100 steps. *Jog: to run at a leisurely, slow pace.*

3. Run 100 steps. *Run: to go quickly by moving the legs more rapidly than at a walk and in such a manner that for an instant in each step both feet are off the ground.*
4. Sprint 100 steps. *Sprint: to run at full speed.*
5. Alternating. Walk 20 steps followed by jogging 20 steps, running 20 steps, sprinting 20 steps, and walking 20 steps.<sup>2</sup>
6. Standing still with the phone for 2 minutes.

The purpose of task 1-5 is generate data the algorithms should analyse and correctly calculate the 100 steps taken. The data from task 6 should be interpreted as 0 steps taken.

These tasks can then be repeated for all carrying positions, different smartphones and multiple persons. In practice the tasks were carried out with two people, the smartphone Samsung Galaxy S III and in one carrying position, strapped to the arm.

After the data collection, we are now ready to test algorithms against each other.

### 6.3.2 Algorithm Selection

To compare algorithms against each other we looked at the precision with which an algorithm could predict the number of steps taken in a data set from a particular task. The algorithm we found to work best was an adaptation of the pedometer in Zhao [5].

**Algorithm Modification** The algorithm in Zhao [5] is for a pedometer working directly with the ADXL345 chip. We, however, rely on a sensor listener on the android platform instead. This means a few differences in our measured data:

- Our data sampling frequency is significantly lower.
- Our data is filtered for noise.

## 6.4 Methodology Observations

In the development of the step counter we

# Test 7

---

## 7.1 Acceptance Test

This section should describe the specification, process, implementation, and results of the acceptance test.

Maybe mention that it is done in collaboration with the TOV course mini project.

## 7.2 Unit Test

This section should describe how we have used unit tests and what purpose, advantages, and disadvantages it has caused. Further it should include observation about unit tests and include a relevant example of a unit test.

### 7.2.1 Reflection for private methods

## 7.3 Test - Reflection

This section is where the reflections about the programmatic part of the test should be. Hence NO reflections about methodology here.

i.e. Unit tests were a good way to allow for refactoring, since it was easy to ensure the functionality was not altered. However, that required extensive unit tests, which we in some cases did not have. This caused some trouble in form of periodic bugs. To solve this we could create a specification of test cases based on the structure of the method. We did not implement this due to lack of feasibility.



# Methodology - Reflection 8

---

## 8.1 Introduction

This entire chapter will contain the reflections and discussions about the methodology XP, as experienced in this project.

## 8.2 40-hour Work Week

## 8.3 Coding Standard

## 8.4 Collective Ownership

## 8.5 Continuous Integration

## 8.6 Metaphor

## 8.7 On-Site Costumer

## 8.8 Pair Programming

## 8.9 Planning

## 8.10 Refactoring

## 8.11 Simple Design

## 8.12 Small Releases

## 8.13 Testing

## 8.14 Other





# Conclusion 9

---

## 9.1 Discussion

## 9.2 Future Work

## 9.3 Conclusion



# Temporary Work Sheets 10

---

## 10.1 Methodology

### Review 1

**12. feb - 20. feb** Normalt vil sprints være 2 kalenderuger. Dette var lidt kortere da vi havde færre forelæsninger i perioden. Indtil 12. feb. brugte vi på opsætning og planlægning - inklusiv projektidéer.

**Vurdering af estimeringer:** Vi estimerer trivielle ting fint. Mere komplekse issues estimeres for lavt grundet manglende ekspertise (vi ved ikke hvordan de skal implementeres), hvilket resulterer i at der er brug for at udføre tidskrævende research, samt åbner op for ikke åbenlyse fejl, der er svære at fikse. Vi brugte 50% længere end estimeret.

**Hvad har vi udrettet?** Vi har lavet de fleste issues - dog mangler vi GUI (#20), database (#16) og song scanner (#17). De tre issues vi ikke nåede blev tilføjet i løbet af sprintet. Equalizer og Playlist er heller ikke lavet, men de var ikke i sprint backloggen.

**Programstatus på master** Programmet på master mangler stadig gui og database, så selvom funktionerne er der, er der ikke nogen måde at benytte dem på. Appen crasher af og til. Der er meget redundant kode (både i test og produktionskode), ubrugte constructors og metoder.

**Ivan:** brug af reactoring?

### Retrospective 1

**12. feb - 20. feb**

#### 10.1.0.1 Coding Standard

We have tried to make the code readable by using field-standards

**Ivan:** def. field-standards?

(e.g. `_privateVariable`). Currently some variable names are too short / un-descriptive. We handle standard conflicts when they are discovered. However a general coding standard has been discussed and decided upon. It was decided that it was not needed to make a specific code standard document.

The coding standard contains decisions about brackets.

**Ivan:** Nesting?, switches?, complexity?

#### 10.1.0.2 Metaphor

Not really relevant for the project because the only people looking at the code are us (the developers) and the supervisor and censor (software people). We instinctively name the classes and methods based on their functionality, so no problems have yet appeared.

#### 10.1.0.3 Refactoring

We have not at this time had the need or reason to refactor.

#### 10.1.0.4 Simple Design

We have very complicated test cases with much replicated code. This is obviously a bad thing, but we contribute this to our inexperience with testing in Android Studio. We have not prioritised this practice, but we found that the production code (not test) was simple and without replication.

#### 10.1.0.5 Pair Programming

We should limit ourselves to using one computer/monitor and stop using teamviewer. Optimally getting an external monitor to put between us as well as a keyboard and mouse. We have to change partners more often than we did so far. We will be better to do all work in pairs, as we until now have solved trivial problem individually. We have found it difficult to solve some problems in pairs, as pair programming assumes the pair knows about/what to program. We have in these cases assigned the problem to one person (e.g. GUI - no one had any knowledge about the problem, so no matter how long the pair would discuss (about nothing), no solution would be viable. Further the problem of discussing thing you don't know about.)

#### 10.1.0.6 Collective Ownership

We have worked with 'collective ownership-like' approaches before, in that the entire group is responsible for all the code at the exam. Before we however used the practice of not editing (or reading) code written by other group members. We will in future sprints incorporate reading the code produced by others and review or refactor it if necessary.

#### 10.1.0.7 Testing

Sometimes we forget to test first. This is bad - we should be more aware of testing first. We should be better at using setup and teardown. We have to focus on having the tests act as a specification. We should not put much effort into "test-to-fail". If it is trivial (boundaries of int, string, etc.) it is okay, but we have to let the tests drive our development rather than hinder it.

#### 10.1.0.8 Continuous Integration

Every time we merge with master we should run all tests and make sure everything runs. Also run the app and make sure it does not crash or have other issues. We have in this sprint solved some assignments right after another without pushing to master or creating new branches (Play/Next/Prev all in one branch because it was easy). We shall be better

to push to master when an assignment is solved and create a new branch for the next assignment.

## Review 2

**23rd February - 6th March** The first day was spent on reading up on XP. We did not label our issues on github with iterations. We should do that (we have the paper issues).

### Estimations:

- We had trouble properly timing our tasks, so we started using everhour on the last day.
- All work related time is tracked.
- We will start counting work hours per person rather than per task.
- We will not use estimations/time trackings from this iteration in our estimation of the next.

### What Did We Accomplish?

- We got stuck for a while on GUI - there were many unpredicted problems.
  - The basic GUI almost works for now, and effort should be greatly reduced, so we can focus on other parts.
  - We should have split this issue up in several subtasks.
- Overall we have a functioning (though unstable) music player.
- About half of the tasks for this iteration were not done.

### Status on Master

- It is possible to get the coverflow out of sync.
- App can crash if next/prev keys are spammed.
- Overall functioning well enough for demonstration purposes.

## Retrospective 2

**23rd February - 6th March**

### 10.1.0.9 Coding Standard

If we have any conflicts we will write it down in a coding standard document. We agreed with Ivan that we basically have a “de facto” (informal) coding standard by working together so long.

### 10.1.0.10 Metaphor

Ivan argued that our problem statement was a sort of metaphor, and we should not discard this practise entirely (as we suggested). The system could also be seen as a pakekeeper and/or a personal trainer.

“pace” might be ambiguous, as it can represent both velocity and SPM.

**Ivan:** def. SPM

#### 10.1.0.11 40-hour Work Week

About halfway in the iteration we found that we would not be able to finish all tasks and we decided to take a few late days (to get just a little bit more done - we did not expect to finish all tasks either way). We found that the productivity was relatively low after hours and the quality of work was so low it was redone the next day. Further we found that the day after was less productive as we were tired. The fact that the next day was lecture and we worked on report (which is very boring) could have influenced to process to the worse. We found that working until dinner was of average productivity, but the productivity and quality seriously dropped after dinner.

##### **Breaks:**

We have experienced that some are too intrigued by the problems at hand that they 'forget' to take a break and stretch their legs. This have led to people 'burning out' before the end of the work day. Another reason to improve breaks is that sometimes people easily gets distracted while researching. It is then important that we We should regard each other, so we do not interrupt a member which are in the zone.

#### 10.1.0.12 Small Releases

We use it, so no comments..

**Ivan:** virker det?

#### 10.1.0.13 On-site Customer

After consideration about the responsibilities of a customer, we decided that Niels (Dan's friend) would not have enough time to fulfill the role. We have therefore decided that we will completely use a simulated costumer.

**Ivan:** surrocate costumer

#### 10.1.0.14 Planning

We have experienced that we have an informal prioritising system, but this can cause problems when choosing new tasks, as these are chosen based on the developers curiosity and interest. The priorities of the tasks should therefore be defined by implementing a stack-like structure in order to ensure the most important tasks are done first.

We had problems with timing our productive work. To solve this we found the time tool Everhour. This will then help us be more precise about estimations. We decided to discard the old estimations and measurements due to their imprecision and to avoid 'muddying' our future estimations and measurements. (Note: we now use the combined time of two developers for estimation and measurement.)

In this iteration we mainly worked on one large task (GUI), which should have been decomposed into many smaller tasks. Further we should be better to create new tasks instead of just added found issues and development to a todo list.

XP makes use of the planning game for estimation of use cases. We first new the game as an individual exercise (explained), but later found, in the planning XP book, a collaborative version was described. We made/make use of planning poker as it has some advantages over the individual version of planning game. We later found that planning

poker was very similar to the collaborative version of the planning game. (REMEMBER TO CHECK SOURCES)

#### 10.1.0.15 Refactoring

We found that our quality is not good enough, so we will in the future (against XP recommendations) have explicit tasks for refactoring.

**Ivan:** see fowler

**Kristian:** refactoring bad code is ok

For now we will assign a number of hours for next iteration, but after that we will write a new issue for refactoring when we discover code smells in areas that are not part of the current issue and these will be estimated and prioritised for the following iteration (maybe they will be fixed as parts of other issues). Code smells in methods relevant to the current issue will be fixed on sight.

#### 10.1.0.16 Simple Design

Same situation as iteration 1. We should look into this maybe probably...

#### 10.1.0.17 Pair Programming

We are no longer using teamviewer. We are working on getting monitors. We should remember the dialogue when working - looking at the driver is not always enough.

Pair programming expects people to know what they are doing - trying out new/unknown code can be difficult. In the future, when in need of research, we will accept breaking with pair programming, whereafter each developer will try things on his own. When a solution is found, the pair will form again and continue from where they left off.

#### 10.1.0.18 Collective Ownership

We are still in the mindset of 'I wrote it, it's my code'. This has resulted in some methods not being refactored because 'It was [name]'s code, I better not touch it'. We will of course try to break this mindset by enforcing refactoring.

#### 10.1.0.19 Testing

We have mainly developed GUI in this iteration, so we have not created many tests, as we have no idea to automatically test GUI. We also already decided to not focus alot on GUI, so we argue that GUI tests are less important.

**Ivan:** no tools?

**Kristian:** we got tools now

There were of course made tests for the changes in dynamic queue and database.

#### 10.1.0.20 Continuous Integration

As mentioned, we have mainly worked on GUI this iteration. This means that we have used the "#20 GUI..." branch as a surrogate master branch. We have done this because



the GUI was not ready to be pushed to master. By assuming the GUI branch was master, we have used continuous integration daily, as we merge and build it multiple times a day. Further we find this practice to be beneficial in larger projects with multiple teams, as we are only one team of 4, we almost never work on more than two branches at the same time.

### 10.1.1 Review 3

**9th March - 20th March** Summary of extended meeting at the bottom of this document.

#### Estimations:

- We started using Everhour to track hours
- We recorded 50 hours (45+5) but counted there would be 168 hours in the iteration. After subtracting time used for stand-up meetings, supervisor meetings, lunch breaks, and small breaks we have about 1 hour per person per day that is not used for anything productive. The time is most likely spent on a combination of general project discussions, switching tasks, and procrastination/not starting when a break ends.
- Some days we did not do anything project related - either because of lack of motivation, illness, or course related stuff (that should have been done at home).
- Our estimation of trivial tasks were approximately twice as long as the actual time.
- We had some tasks take much longer time than estimated, given that we ran into problems with a library.
- In the future we will talk about possible solutions for each task in order to estimate risks to according to possible problems.

#### What Did We Accomplish?

- We made a music library and a working test suite, further we made it possible to test private methods.
- Made a configuration table
- Wrote report.

#### Status on Master

- Some song scanner capabilities are added.
- App can crash (it always could, but it needs to stop.)

#### 10.1.1.1 Extended Meeting

**Minimum Viable Product** Antagelser:

- 70 timers arbejde per iteration. Dette svarer til ca. 20 units.

**Ivan:** omsk. velocity begreb

- Der skal skrives en rapport.
- Der skal laves en app.

MVP:

- Appen skulle kunne tælle skridt.
- Appen skulle kunne tilføje sange til sangbiblioteket fra en brugervalgt mappe eller standard mappen (Music).
- Appen skulle kunne matche en sang til et tempo.
- Appen skulle kunne kontrolleres til at kunne: Play, Stop, Pause, Next, Previous. Uden brug af en tændt skærm.
- Appen skulle kunne afspille musik og hvad der dertil tilhører (skift til næste efter slut, etc.).
- Det er antaget at MVP er funktionel fra armen.
- Appen skal være gennemtestet!

**Ivan:** stort krav

- Appen skal kunne automatisk hente BPM data om en sang fra internettet.

### **Protocols: New Issues, Switching Issues, Changing Issues During an Iteration**

Snak og undersøg om det er 'lovlig' at skifte opgave uden den igangværende er færdig. Snak og undersøg om det er 'lovlig' at ændre på issues der er aftalt (samt hvordan dette kommunikeres effektivt).

If an issue (without any relation to existing issues) is discovered:

- If it is very important, discuss it amongst the group and decide whether it should replace an existing issue.
- If it is not important, the issue is added to the next iteration planning.

When choosing a new task to work on, it is important to select tasks in progress, if any. This can be understood as issues in progress are prioritised highest. Remember it is okay to 'take' tasks from other members if they do not work on the task assigned to them.

If (part of) an issue is deemed not important to the project, it should be discussed and agreed between ALL members of the team. If not all members are present, the missing members are contacted to set of a meeting. If no response, then pause the issue and discuss it when response.

**Ivan:** ??

**Pushing to Master and Fixing Crashes** How do we make sure the maser is in a good place.

Fix master (crash without test files). It is very important that we test more comprehensive tests, e.g. input null, "", -1, file exists. Further it is important we check to see if the tests are comprehensive before we push to master.

ps. read about practise of deleting branches.

**Releases** Aftal dato for næste release.

Release should contain:

- See Software Innovation 3 - Configuration Table
- Step Counter + Music Player + Song Scanner
- 10 April. (next iteration end)

**Report** Tilpas konfigurationstabel. Lav toc.

**Architecture** Modularisering af projektet (med interfaces?)

We will refactor and create an architecture after release.

## Retrospective 3

**9th March - 20th March**

### 10.1.1.2 Coding Standard

In methods it is done as:

```
1 if (statement) {  
2     //Do stuff  
3 }  
4 else {  
5     //Do stuff  
6 }
```

Listing 10.1: Coding standard for statements and loops.

Trivial getters and setters (or other methods that simply do a return) should be on one line.

Use long variable names please. So no ‘am’ for AudioManager, use ‘audioManager’ as variable name.

### 10.1.1.3 Metaphor

Nothing in particular for this.

### 10.1.1.4 40-hour Work Week

We have encountered sickness, so we have not even reached the 40 hours.

### 10.1.1.5 Small Releases

We have decided to ignore our earlier “release” in order to plan a real release at the end of the next iteration (10. april).

**10.1.1.6 On-site Customer**

We have made preparations for us to handle the On-site Customer role by simulation.

**10.1.1.7 Planning**

We don't see the point of individual estimations as individuals rarely finish issues alone.

If an issue (without any relation to existing issues) is discovered:

- If it is very important, discuss it amongst the group and decide whether it should replace an existing issue. The issue is then estimated by the pair handling it.
- If it is not important, the issue is added to the next iteration planning.

If an issue (with relation to existing issues) is discovered:

- It should be estimated by the pair which handles it.

**10.1.1.8 Refactoring**

Given that we will be examined in the code, we have decided to refactor more than suggested by XP. Further we have decided to refactor and make a new architecture after the release. This architecture should be of a simple design. See simple design.

**10.1.1.9 Simple Design**

We have found that the individual 'modules' in our code are not strongly defined. We plan on solving this by implementing each module through interfaces - thereby giving a clear overview of the public methods for each class and the inputs and outputs for each class and method. This will improve the independence of each module making it possible to change an entire module without affecting the overall program.

**10.1.1.10 Pair Programming**

We found that the forced timed pair programming switch didn't sit right with us. We experience multiple times that we should switch just before the current issue was done. This created a lot of overhead. We therefore decided to use a task-based approach where switches only are made between issues or between issues estimated to take more than 3 hours.

We have used the practice of, when in problems, splitting the pair, where each partner then makes some prototypes to solve the problems. Then the pair reunites and solves the problem together.

We have had some sickness this iteration. This meant that we have not been pair programming when the members were working from home. This was not a problem since the tasks solved by individuals were sufficiently trivial. This gives rise to the question of when a task is trivial enough to not pair program.

**10.1.1.11 Collective Ownership**

We are still stuck of the old method of blaming others and being defensive of own work. We will once again try to better ourselves.

**10.1.1.12 Testing**

We found our previous testing method was insufficient in regards to private methods and we changed the method so private methods are now tested. Also we need to be more aware of testing first, as we use TDD. We need to be more comprehensive when testing, and not only sticking to a specification approach. e.g. input null, “”, -1, file exists.

For boundary tests as input it is okay to test them by creating an array with all the desired values, whereafter an loop iterates over and calling the method with all the values.

**10.1.1.13 Continuous Integration**

We have had some problems making sure the master branch is stable. See the extended meeting from Iteration Review 3 (Pushing to Master and Fixing Crashes). This has been a problem all the time, but the issue was not caught until now.

### 10.1.2 Review 4

**23rd March - 10th April** Summary of extended meeting at the bottom of this document.

#### Estimations:

- We expected to work 70 hours, 67 hours on code and 3 hours on report. It turned out we worked 71 hours in total, 68 hours on code and 3 hours on report. This estimation is concluded to be spot on, as the measurements have small variations, which can make up for the extra hour.
- We have solved issues for 82 estimated hours in this iteration (non report only, nor extra bug fixed (4 hours)).
- Based on estimations of the issues of the iteration, we can conclude that when an issue is estimated between 8 (10) and 20 hours, the time actually used is approximately 60% of the estimated time. However when we estimate non-trivial short issues (2-4 hours) we estimate fairly low in comparison to the actual used time. i.e. two issues estimated to 2 hours took 8 hours and 3,5 hours respectively.
- The small deviation in time between estimated and actual used time, can be traced to the granularity of our estimations. We have a granularity where we estimate in actual hours in up to 2 hours. More than 2 hours are estimated in units of half a day. 4 hours are half a day, 8 hours are an entire day, 10 hours are a day and a half ... We used these numbers because of the restrictions of the online planning poker tool used. This actually works okay, as it gives the opportunity to estimate trivial issues to less than half a day, which obviously will give a lot of overhead.

**Ivan:** Good reflections and good section.

#### What Did We Accomplish?

- We finished the step counter
- We fixed the out-of-memory issue with the album covers.
- We created the GUI for settings and SongScanner.
- We fixed a number of small bugs.
  - Next song after completion.
  - Handle missing songs.
  - Stop button skipped to next song.
  - Set filename as song title if no other title is available from the MP3 file itself
- We now only need the following to finish our MVP
  - Acceptance tests
  - Control without looking at screen
  - Finish BPM online tests
  - General unit tests

- Settings for the step counter, i.e. sensitivity
- Refactoring. Modulation, and correct encapsulation.

### Status on Master

- Working and tested
  - Step Counter
  - Music Player
  - Song Scanner
- Issues still on master
  - Step Counter
    - \* Works as intended
  - Music Player
    - \* Invalid state exceptions still happen on master (warnings in log)
    - \* prev and next pauses song when playing, it should continue playing the new song.
    - \* Seekbar seekTo functions correctly, but have display issues jumps when dragging.
  - Song Scanner
    - \* Missing online BPM lookup in Song Scanner.
    - \* Song Scanner Settings not used in app.

## Retrospective 4

23rd March - 10th April

### 10.1.2.1 Coding Standard

In this iteration we decided to postpone the decisions and documentation of the coding standards to next iteration, because we intend to allocate time for refactoring next iteration.

### 10.1.2.2 Metaphor

Given the size of the project and the fact we all have worked together before, we are generally on track with the unspoken metaphor, which is essentially just what the program is. i.e. the music player is called music player. Besides the small team makes it easy to communicate if and when conflicts of understanding occur. All this makes the metaphor an implicit understanding between the team members.

**Ivan:** Title is a type of metaphor - reflect upon this. Maybe even change the title.

**Ivan:** Icons are types of metaphors - reflect upon this. Maybe even change the icons.



### 10.1.2.3 40-hour Work Week

Besides single and uncorrelated episodes, we have not had any problems with energy. Some of this energy can also be attributed to the fact we have mostly coded and not written much report this iteration.

We suspect this situation can change in the coming iteration, due to the fact we are going to write a lot of report, which is boring and tiring. We will reflect upon the results in the next iteration retrospective.

### 10.1.2.4 Small Releases

The end of this iteration is our first real release. It is not a small release, but the app was not in a “releasable” state previously. We plan on making smaller releases from this point on.

We do not have any actual customers to show the release to. This have been a factor in the slow release.

Although we have no customer to test and evaluate our releases, we still benefit from doing them. First off, the small release forces us to merge and keep the master up-to-date.

Every merge to master is accompanied with making all test pass. The requirement of small release forces frequents merges to master, resulting in up to date (all the new features) version of the app.

Further, using small releases gives greater incentive for the team to develop and complete concrete, delimited functionality as requested or agreed with the customer. i.e. If the customer wants a feature (pacer), this feature (pacer) is developed as a separate module to be changed/improved later without corrupting other modules. The feature is also completed, if possible, before release, so it is not going to dangle as otherwise could happen.

### 10.1.2.5 On-site Customer

**Ivan:** Haven't we used a surrogate customer?

**What has the absence of a customer meant for our project?** Because we have not properly used a surrogate customer, we have not discussed the project as much as we could have (feedback), and we have not gotten any acceptance tests done. Overall this has not been a big issue, because we have been on track - moving towards finishing our MVP.

**How have we dealt with the assignments normally dedicated to the customer?**  
**Prioritising:**

We have prioritised use cases sorted by new interesting non-trivial features (novelty), whereas a customer might sort use cases by price, usability, and value (cost/value). This is because we do not gain much from implementing trivial features or nearly identical to existing features.

**Acceptance Tests:**

We have yet to make any acceptance tests, so this assignment has not been filled.

**User stories / Use Cases:**

We have not used user stories, instead we have used issues with a small explanation of the functionality. This is a bit like user stories, but not entirely. This has worked fine, as we only have ourselves to answer to.

**Minimum Viable Product:**

We decided on the MVP based on requirements from the study regulations and our own interests based on novelty.

**Have we made decisions or changes which could not have been done by a customer in real life?** We decided not to include the interval trainer because it would be trivial to implement, given its similarity to the already existing pacer. A real customer would might have wanted this interval trainer features, because of its high value and low cost. As explained, we prioritised based on novelty and not cost/value, hence this decision.

**10.1.2.6 Planning**

We estimated 70 hours of work for this iteration, which is a bit more than we spent last iteration, but since we had some problems with people being ill last iteration, we decided to assume we would spend more time, which turned out to be very accurate, as we used 71 hours. We made some notes on planning in iteration review 4 and 5.

**10.1.2.7 Refactoring**

Not much energy was spent on refactoring - most things were spent on new functionality - because next iteration is going to be dedicated to refactoring. Minor bad smells were corrected.

We plan on making a thorough refactoring plan, for the next iteration, based on [3].

**10.1.2.8 Simple Design**

Simple design has been followed fairly well in this iteration, but because of the lack of refactoring some parts of the program are a little more messy than they should be - the plan is to fix this in the next iteration.

**10.1.2.9 Pair Programming**

After setting up monitors and keyboards our pair programming has improved. Sometimes trivial/small tasks were solved by single persons, but most of the time pair programming is used. Some of us have a tendency to forget changing drivers often, and while we could change partners more often we have done so fairly regularly. We changed partners when it felt natural rather than on set times.

**Why do we change pairs with each issue?** We experienced that changing from one issue to the next before completing the first, was awkward. This can however decrease the feeling of collective ownership, hence the developer will come in less contact with each implementation.

**Why did the other way not work?** We experienced that when changing issues, the developers flow could be broken, which would result in wasted (overhead) time.

**Is there an overhead when using Pair Programming as much as we do? When should we use Pair Programming? How do we minimise the overhead?** Both Beck (find source) and our experience agree Pair Programming results in overhead. However this overhead should be minimised as much as possible. We have therefore decided to not pair program when implementing trivial issues. The triviality of issues is determined by the individual or pair in the process of programming.

**Should we change drivers often?** We do not think we need a fixed time frame one can be driver, sometimes it makes sense to drive for a long period of time if one possesses a relatively better understanding of the issue. i.e. As Becks example where a experienced person is paired with a inexperienced person. First the inexperienced person observes and learns what is going on, later the inexperienced person becomes experienced and can contribute and drive. This scenario can occur when changing partners between issues.

In turn, it also makes sense to switch often when there is a similar level of understanding in the pair since it ensures ideas from both programmers will be heard.

**What impact has it had not doing so?** The idea behind changing drivers is that it enforces collective code ownership (really?). This, however, requires that partner switch is done appropriately so that one person does not remain on the issue all the time.

#### 10.1.2.10 Collective Ownership

Collective ownership has not influenced our project a lot in this iteration, but we have gotten a stronger feeling of the code actually being collectively owned - there is not much feeling of something being someone's code.

**We still assign issues based on who have done what - e.g. Kristian writes settings GUI, it is his GUI.** By letting an experienced person work with a less experienced person does not require overhead, we reduce the amount of overhead related to having to learn the new worker. It also prevents some errors/misunderstandings associated with having to "learn" the code. We have a tendency to have one person always stuck on the issue, and alternating the partners, instead of letting the first "partner" team up with a different person. This allows the person to follow his idea for a solution to the end, but also makes it feel more like "his" code.

#### 10.1.2.11 Testing

As always we have not been testing the GUI. Our tests have gotten fairly big, and therefore they are beginning to take a while to run (could be solved by using an integration server, see Continuous Integration).

We have had some trouble remembering to do test first - especially when our methods get complicated. In those cases we have sometimes written the tests after finishing the functional code. This has especially been the case when we were unsure what we needed to test (i.e., when we did not know how the method was supposed to work). Basically we ended up using spikes, but instead of throwing out the code we ended up using it and writing tests.

**10.1.2.12 Continuous Integration**

We are still doing as we did to begin with: Merge with master and run tests. But the tests are starting to take a long time to run. Ideally we want a dedicated integration server, but it is not realistic for us to set one up at this point. In the end we will have to be more selective with when we run our tests.

**Is this way of doing it viable? How long is it viable? Why?** This way of doing it would not be viable for a larger project, but the cost of setting up an automated build server for a project of this size would most likely be bigger than sticking to the manual approach.

**What impact has this way of doing it had?** We get very frustrated when shit doesn't work, because it takes forever to do over.

We are experiencing some intermittent exception from an unknown source. This would have been caught by an automated test tool, at least given us a log of when and where it first appeared. Instead we are now using time debugging everything, this had of course not been acceptable in a larger project.

### 10.1.3 Review 5

#### 13th April - 24th April Plan:

- Use the length for each unit in hours instead of the hours used in planning poker.
- Based on experience, 70 hours an an itera of work each iteration. continue to plan 70 hours of work each iteration. continue to plan 70 hours of work each iteration.

Estimated Planning Poker Time	Time Allocated in Project and Github
0 hours	N/A
0.5 hours	N/A
1 hours	1 hour (min estimate, trivial)
2 hours	N/A
3 hours	1 Unit - Half a day - 3 hours
5 hours	N/A
8 hours	2 Units - a day - 6 hours
13 hours	3 Units - a day and a half - 9 hours
20 hours	4 Units - two days - 12 hours
40 hours	5 Units - two and a half days - 15 hours
100 hours	6 Units - three days - 18 hours
??	Unknown

Summary of extended meeting at the bottom of this document.

#### Estimations:

- We did not really estimate anything this iteration.
- We forgot to track the time used for most of the iteration. This was because we refactored loosely and did not create issues for all assignments.
- We used a lot of time on review of retrospective of last iteration, this should be tracked and estimated as well.

#### What Did We Accomplish?

- Overall this iteration was much less productive than planned - we expected the more "boring" tasks to result in less productivity, but we were unable to handle it in a satisfying way.
- We made a refactoring plan, ensuring somewhat similar refactoring patterns.
- We refactored a small part of the program.
- We fixed (worked around) a periodic bug because the cover flow hangs, which we solved by just turning off the screen.
- We discovered that we have not quite tested well enough.
- All in all we have been VERY VERY unproductive and have been too easily distracted. It has been bad enough so the other group has been more productive than us.

## Status on Master

- Some issues were resolved on master
  - online BPM
  - periodic crash doing tests
- Issues still on master
  - Music Player
    - \* prev and next pauses song when playing, it should continue playing the new song.
    - \* Seekbar seekTo functions correctly, but have display issues jumps when dragging.
  - Song Scanner
    - \* Song Scanner Settings not used in app.
  - Tests are incomplete
    - \* BPM in DB not tested

## Retrospective 5

### 13th April - 24th April

#### 10.1.3.1 Coding Standard

We have expanded our coding standard by writing down the general structure and bad smells. Otherwise we have used verbally agreed upon standards.

It has become clear this would have benefited us much earlier in the project, essentially saving much of the time spend this iteration on refactoring. Though some of the considerations done to reach this structure, was not possible to consider before encountering the problem. However, we now this is a good structure, hence it can be used from the get-go in future projects.

#### 10.1.3.2 Metaphor

We discussed metaphor at the supervisor meeting. It was noted that both the program title and program icon can be seen as metaphors. This is due to the fact the title and icon should illustrate the purpose and functionality of the program.

We have not made any changes to our previous title or icon, but upon reflection it is likely we change the title in the future.

We have encountered several issues when writing the report. Many of these issues, and the difficulty getting started, can be contributed to the missing metaphor or vision for the report. Contrary to the program, we do not share, or have discussed, any common metaphor of the report

### 10.1.3.3 40-hour Work Week

The 40-hour work week practice is used to combat burnout. In this iteration we clearly worked much less than 40 hours a week, however, we have still experienced burnout.

One major culprit is when one is not sure about what to write (typically report) or what to do. Last retrospective we assumed the lack of energy when writing report was solely due to lack of interest. Now we believe it is both because of the lack of interest, lack of shared vision, and bit because of our pre-assumed idea of the report writing being boring.

Another reason for our burnout is the lack of shared vision as explained in metaphor last paragraph.

Further we did not plan thoroughly, hence a lot of work needed to be done before actually starting on an issue. This has caused many to be less enthusiastic and just not start working on any issues.

This has also shown us how important planning and structure really is.

### 10.1.3.4 Small Releases

Although creating a release last iteration, we have not used it for anything at all since. This is due to the fact we have not gotten any external testers or customers to test it.

The reason for us not taking the role as surrogate customer and testing it, is we have tested the shit out of it when creating it and every time we debugged and ran it, hence we did already know how and where it works. This leads to a very low usability, given that novelty is prioritised higher. The scale is very much tipped in favor of novelty, given we have no user to complain. This decision is not made actively, but because we are heading for an exam and not actually distributing the program, we have experienced that novelty is rated higher. Furthermore, the only usages of the program is done by us at the demonstration, hence the low limit of usability.

If we should have followed XP to the letter, we should have taken the time to (user) test it thoroughly as a customer.

### 10.1.3.5 On-site Customer

When taking the role as on-site customer, we have a tendency of polluting our decisions based on what is interesting to develop. We, as a surrogate customer, focus more on what is interesting for our exam than on things that may be interesting for an on-site customer.

e.g. Usability issues are prioritised low.

The acceptance tests, we should create as a surrogate customer, will be done as a mini project in TOV. The reason for us to focus on acceptance test in TOV, is because we cannot reflect upon it before trying it, and it is an important part of testing in XP.

### 10.1.3.6 Planning

**Time Tracking:** We started tracking time for a couple of iterations ago.

**Why time track?** The initial reasons for time tracking was estimation improvement and code velocity. Since then we have tried to use the metrics for measuring actual work hours and conclude and improve upon these, if possible.

**What is relevant to time track?** We have discussed whether we should time track meetings and the likes. On one hand these meetings can be used to measure actual work hours, hence improving the general working speed. On the other hand it may cause more overhead without any insurance it actually can improve general working speed.

A thing worth noticing is each second is time tracked, but this can be misleading and lead to nitpicking instead of what is important. To counter this, Bech through XP, suggest to measure in units (of half a day). This is however only recommended to code issues.

**Big Brother vs. Anarchy** Since we started tracking time we have found more and more things we wanted to track. This can result in more overhead (actual “real time” and mental), and more importantly in having a work environment where Big Brother “Everhour” watches your every step if we do not set a line.

The problem with logging everything is how conclusions are drawn. If one concludes that the best programmer is based on “Lines of code per minute” or similar, is that it can be very misleading. In this case there is no thought about quality, hence the real best programmer might be assessed to be a lousy programmer.

Some members forget or didn’t bother to start the time tracking tool. This could be caused by laziness or because they felt it was not really necessary. This could also be due to the fact we are not used to tracking our time with a tool.

The bottom line is that time tracking should improve our process.

**Planning:** We decided not to estimate or create issues for the refactoring and report process, due to not knowing what was to be done. As explained above, this decision was cause for some problems with concentration and energy. In the future we will therefore create issues for all processes and estimate all code issues, to counter this problem.

In summation: We should actually plan an iteration before doing it.

#### 10.1.3.7 Refactoring

We refactored as an activity this iteration in spite of what XP tells us.

We created a refactoring plan to ensure consistency across all classes and coders.

We did this because of the reasons stated in retrospective 4.

We did not refactor as much as planned, but we got a clear idea how to increase the quality of all our code.

We ended up doing some things that could be considered functionality while refactoring, e.g. we removed stubs and connected some unconnected parts of the code (that should be connected). This made sense for us, however, as it clears up almost finished functionality without making big changes to functionality.



#### 10.1.3.8 Simple Design

Since we have not really developed anything new, we have not used this a lot.

We have removed a few things that were not “simple design” when refactoring, but it seems like we have not been fully aware of what “simple design” actually means.

“Put in what you need when you need it.”

#### 10.1.3.9 Pair Programming

We are not sure when we gain something from pair programming - some trivial code could be refactored by one person, but maybe doing it in pairs will give a better end result. We will experiment with different settings in the last few iterations.

note: only relevant due to we refactor as an activity.

#### 10.1.3.10 Collective Ownership

Refactoring as an activity has improved our collective ownership, as more people get to go over the code that has been written by others.

Refactoring, while implementing, code written by other pairs would have the same effect, but we noticed that people often steer away from changing code that was not written by themselves.

In conclusion: Collective ownership is hard, but gets better over time.

#### 10.1.3.11 Testing

**Adding test case after finding a bug, why is this not done?** We did this to a small extent but we are not consistent with it and at this point we might as well not. Since the extra quality we get from writing a test for this does not justify the time it takes. We would gain experience with testing against bugs, but we do not learn much about process, increase the quality of the rest of our code nor benefit from it at the exam, since very few supervisors and censors read it.

**We should evaluate the coverage of our test:** We should remember to compare our coverage with how many bugs we actually find. Find appropriate methods from test and verification course.

**Needed tests were not made, discuss!** We found bugs that should have been caught by tests (such as does this feature even do anything)

Because we have problems writing a sufficient amount of tests, we should figure out a solution that could solve this. Because of time we will most likely not implement the solution. We could supply some test templates/specifications, telling us how to test certain types of methods, i.e., all methods with parameters should at least have a boundary test.

We will not specify test templates at this stage of the project, but we will review our current tests and discern if any is missing.

The specific cases of missing test cases will be examined to determine if they actually are missing or if there is a test, which does not test correctly.

**10.1.3.12 Continuous Integration**

We still have the same issue of tests taking a long time, but after working around some trouble with the UI it is now significantly faster, and the lack of integration server is not as big an issue as before.

## 10.2 Refactoring

### Song - Class

We refactored the song class and we found these bad smells.

1. Constants was written without ‘\_’ as whitespace. Was fixed by inserting ‘\_’ as whitespace.  
e.g. MS\_PER\_SEC
2. We have restructured the class into five sections. This is done because we had much trouble determining which methods were public accessible and which was wrongly made public. Due to this we named the five sections:
  - a) Private Shared Resources
  - b) Accessors
  - c) Constructors
  - d) Private Functionality
  - e) Public Functionality - Interface

Other classes may still have a section called ‘Stubs and Drivers’ in the top of the class. Each section is encapsulated with a ‘//region’ and ‘//endregion’ making it collapsible. e.g.

```
1  //////////////////////////////////////  
2  //                               Private Shared Resources           //  
3  //////////////////////////////////////  
4  //region  
5      Code...  
6  //endregion
```

Listing 10.2: Example of code section.

3. Refactored the method `getDurationInMinAndSec` to use ternary operators. This reduced 6 lines of nearly identical if-statements to lines using ternary operators.
4. In `getDurationInMinAndSec` we found that the method used the class’ own getter to access the variable `_durationInSec`. This has been fixed by accessing the variable directly.



# Appendix



# Releases - (After Methodology)

---

## A.1 Introduction

Mention how this is part of **Essence** and not **eXtreme Programming**.

Maybe even mention that this was made as part of SWI course mini project?

## A.2 1st Release

This section should contain the configuration table and description for the 1st release.

## A.3 2nd Release

This section should contain the configuration table and description for the 2nd release.





# Tactile Interface - (After Step Counter)

---



## B.1 Use Cases

This section should describe the use cases set up for the Music Player feature.

i.e. The Music Player should play the next song after completion of the current playing song.

## B.2 Implementation

This section should describe the programmatically interesting and relevant parts of the Music Player.

i.e. We found that trying to stop the music player while not loaded caused it to skip to next song. It was due to the fact this was an illegal action in that state, as seen in Figure (Android graph over mediaplayer states). We fixed this issue by implementing something, as seen in Listing (Code snippet of interesting code).

## B.3 Methodology Observations

This section should contain the observation about the XP methodology gathered during the development of the Music Player.

i.e. When developing the music player we have trouble adhering to the **Simple Design** principle, which states “Only develop what is need now”, because we know we will implement additional features later, which lead us to implement methods for future use.



# Bibliography

---

- [1] Google Android. Sensors overview. [http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html), 2015. [Online; Accessed 21-04-2015].
- [2] Judy Edworthy and Hannah Waring. The effects of music tempo and loudness level on treadmill exercise. *Ergonomics*, 49(15):1597–1610, 2006. doi: 10.1080/00140130600899104. URL <http://dx.doi.org/10.1080/00140130600899104>. PMID: 17090506.
- [3] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., 1st edition, 1999. ISBN 0-201-48567-2.
- [4] T. Yamamoto, T. Ohkuwa, H. Itoh, M. Kitoh, J. Terasawa, T. Tsuda, S. Kitagawa, and Y. Sato. Effects of pre-exercise listening to slow and fast rhythm music on supra-maximal cycle performance and selected metabolic variables. *Archives Of Physiology And Biochemistry*, 111(3):211–214, 2003. doi: 10.1076/apab.111.3.211.23464. URL <http://dx.doi.org/10.1076/apab.111.3.211.23464>. PMID: 14972741.
- [5] Neil Zhao. Full-featured pedometer design realized with 3-axis digital accelerometer. <http://www.analog.com/media/en/technical-documentation/analog-dialogue/pedometer.pdf>, 2010. [Online; Accessed 21-04-2015].



# Project CD

---

The CD found on this page contains the following:

- The source code for
- A compiled version of
- A digital version of the report in PDF format.



# Examples & ToDo D

---

**Alexander:** Example of comment/ToDo made by Alexander

**Christoffer:** Example of comment/ToDo made by Christoffer

**Dan:** Example of comment/ToDo made by Dan

**Kristian:** Example of comment/ToDo made by Kristian

**Ivan:** Example of comment/ToDo made by Ivan

```
1 public class HelloWorld {  
2  
3     public static void main(String[] args) {  
4         System.out.println("Hello, World");  
5     }  
6  
7 }
```

Listing D.1: Caption of code snippet

This is how you refer to a source written by Edworthy and Waring [2].

## List of Todos

- i,  **Ivan:** Metaphor - Think about more suitable title.
- 1,  **Alexander:** BPM error deviation = 7.5 BPM - <http://www.cs.virginia.edu/~stankovic/psfiles/MusicalHeart-SenSys2012-CameraReady.pdf>
- 1,  **Ivan:** Write about XP
- 2,  **Ivan:** Deliberate on what Yamamoto et al. [4] is about.
- 11,  **Christoffer:** Intro omskrives?
- 19,  **Ivan:** brug af reactoring?
- 19,  **Ivan:** def. field-standards?
- 19,  **Ivan:** Nesting?, switches?, complexity?
- 22,  **Ivan:** def. SPM
- 23,  **Ivan:** virker det?
- 23,  **Ivan:** surrogate costumer
- 24,  **Ivan:** see fowler
- 24,  **Kristian:** refactoring bad code is ok
- 24,  **Ivan:** no tools?
- 24,  **Kristian:** we got tools now
- 26,  **Ivan:** omsk. velocity begreb
- 27,  **Ivan:** stort krav
- 27,  **Ivan:** ??
- 31,  **Ivan:** Good reflections and good section.
- 32,  **Ivan:** Title is a type of metaphor - reflect upon this. Maybe even change the title.
- 32,  **Ivan:** Icons are types of metaphors - reflect upon this. Maybe even change the icons.
- 33,  **Ivan:** Haven't we used a surrogate customer?
- 55,  **Alexander:** Example of comment/ToDo made by Alexander
- 55,  **Christoffer:** Example of comment/ToDo made by Christoffer
- 55,  **Dan:** Example of comment/ToDo made by Dan
- 55,  **Kristian:** Example of comment/ToDo made by Kristian
- 55,  **Ivan:** Example of comment/ToDo made by Ivan