Business Academy Aarhus

Aarhus BSS CobeLab

Problem Statement

CobeLab Experiment Software

Author:
Iustinian Olaru

Supervisor:
Prof. Eryk Dybdal

March 8, 2016

Contents

1 Contact page

COMPANY

Name :	Cognition and Behavior Laboratory, Aarhus BSS
Address :	Nordre Ringgade 1, DK-8000 Aarhus C
Phone :	+45 8715 0000
Fax :	+45 8715 0201
CVR :	31119103
E-mail :	bss@au.dk
URL :	bss.au.dk/

LAB MANAGER

Name :	Dan Mønster
Office :	bldg. 2622, 10
E-mail :	danm@econ.au.dk
Mobile :	+45 93 50 80 55

ASSISTANT LAB MANAGER

Name :	Joshua Brain
Office :	bldg. 2627, 9
E-mail:	jbrain@econ.au.dk
Phone :	+45 87 16 61 52

SOFTWARE DEVELOPER INTERN

Name :	Iustinian Olaru
Class :	DMU13-i
E-mail :	o.iustin@gmail.com
Phone :	+45 23 49 63 21

2 Project description

2.1 Project Background

The project is being developed under the supervision of Dan Mønster, Assistant Professor at the Department of Economics and Business at Aarhus University. Apart from Dan , the project has two other researchers, namely Assistant Professor Riccardo Fusaroli PhD, Associate Professor Kristian Tylen The three researchers are currently planning to conduct an experiment starting December 2015 that aims to investigate heart rate synchronization in social interactions.

"Several recent studies have shown that people's heart rates synchronize when they engage in social interactions. However, the mechanisms behind this phenomenon are unclear. The purpose of our study is to investigate several proposed mechanisms. It is well known that heart rate is influenced by respiration (respiratory sinus arrhythmia) and movement. Both respiration and movement play roles in coordination during social interactions. Conversations, for instance involve complementary respiration due to turn taking and both synchronous (postural sways) and complementary (gesture) movement. The main purpose of our study is to detail the role of synchronous and complementary speech and gesture on heart rate coordination, in the context of both controlled and more naturalistic interactions." (Dan Mønster, Kristian Tylen , Riccardo Fusaroli (2014))

In this project, I will be working on an application that I started in the internship called CobeLab Experiment Software (C.O.B.E.S.). The purpose of the internship was to experiment with different architecture types and different third party libraries in order to determine which one suits our needs best. Although the application is already started, it is by no means near completion.

The C.O.B.E.S. application developed in the internship already has some features implemented such as audio input and recording, Wii remote input and interaction, and also an existing graphical interface for the implemented features. My intention for the Main Project is to rewrite and enhance most of these features because the implementation I have made does not support certain features, or does not allow the program to perform at specific quality standards, such as high frame rate and fluent graphical feedback, which are essential for the success of the project and the experiment. Also, because the initial application was a learning project, there are some parts which are not needed for the final solution, or that have been implemented in more than one way in order to determine the best possible solution. Such sections of code will be removed from the project also.

2.2 Project Purpose

The purpose of this project is to design and develop the application started in the internship period and to implement the software solution that will allow the researchers at Cognition and Behavior Lab Aarhus University to conduct their experiment and gather research data. The software will support the input of various hardware such as Wii remotes, microphones to record audio input. It will also present different types of stimuli for the test subjects to interact with . The software will also communicate with a third party physiological recording system called BIOPAC that will record physical measures from a participant's body (i.e. heart rate , breathing pattern).

Participants of the experiment will be presented with two types of stimuli: synchronous and asynchronous. The asynchronous task will have the test subjects play a game in which each of them will be assigned a pointer on the screen that will be controlled with a Wii remote. The participants will have to one of the three types of courses that will be randomly selected for each session. The course types are: 1. Keep the pointers inside a moving circle; 2. Go through a maze type course where the course will be gradually revealed as they advance further; 3. Traverse a relay type course where participants will take turns in traversing the course and waiting for the partner to reach them so that they continue the course.

The purpose of the project is to present these stimuli to the participants and allow the researchers to control the speed and tempo of the tasks so that relevant data can be obtained from the experiment.

2.3 Expected result

2.3.1 Reports

The first expected result of this project will be a Main Project Report centred around the whole project and different development stages. I'm expecting the report to contain: descriptions of the whole development process, design, and possibly deployment if the research team decides to have the trial period during my period here at the laboratory.

If I have enough time left in my project period I also plan on looking into several alternative methods to develop this software solution. One such alternative could be to use the Unity game engine to have a certain user experience that would make it enjoyable for both the experiment conductor and also the participants.

2.3.2 Programs

The project will focus on building a program that can implement all the necessary functionality in order to facilitate the execution of experiments for the aforementioned study.

I am expecting to implement a software solution with a proper design architecture that will allow the program to function ideally. The design of the program should be easy to read as other developers might want to develop the software further. One of the future possibilities for the software was to modularizes the whole program so that different types of data can be gathered, different types of inputs can be used and different types of stimuli can be presented to experiments participants. If I have time, I would like to look into different possibilities and ideas that might be used to make it possible and develop this software in that direction.

The program is expected to have fully functional input from Wii remotes and microphones and is able to communicate with the third-party application BIOPAC. BIOPAC hardware and commensurate software allows the recording of various body functions. In this experiment, the BIOPAC will record breathing patterns and heart rate signals. The software I am developing is expected to communicate with the BIOPAC and inform it whenever a certain event has happened in the experiment so that time stamps can be recorded along with the signals of heart rate and breathing patterns.

I am also expecting that the User Experience of the software will be quite well made so that experiment participants do not have a hard time interacting with the software and that it provides a pleasant experience of interaction between the two participants. The user experience will be very important when presenting the various experiment stimuli to the participants.

In the experiment, the participants are going to perform a task synchronously which will require each of them to control a dot on the screen with a Wii remote and remain within the boundaries of a moving circle on the screen. It is therefore important that the graphical aspect of the program performs smoothly as to prevent participant frustrations which would confound the experiment data.

Another such example where the same problems can cause the same effect are in the synchronous stimuli window. In this window, the participants will be presented with a text to read at the same time. The text will be highlighted so that a reading rhythm is established by the experiment controller. If the text highlight is too fast or to slow the participants might have the same feeling of frustration.

For the stimuli part of the experiment, I am expecting that in the finished product all three course types will be available to be played and that the asynchronous task window will have a proper UX that will make the reading task easy to follow for the users. Also I expect to set up the whole user experience for the experiment conductors so that the whole flow of the experiment will be automated and handled by the software in the way the project owners expect.

The program will have the following functionality features:

- Wiimote interaction and feedback;
- Microphone implementation and feedback;
- Text prompter;
- Interaction and communication between application and BIOPAC;
- Data recording of device values(Wiimote accelerator values, Wiimote infrared detection values, microphone recording and feedback);

- Graphical stimuli for participants;
- Graphical Feedback and general responsiveness for researchers;
- Good quality User Experience;

3 Company description

Cognition and Behavior Lab Aarhus University is a research facility that functions under strict scientific and ethical regulations under the Faculty of Business and Social Sciences. The laboratory was founded in 2013 with Dan Mønster as Lab Manager and Joshua Brain as Assistant Lab Manager .

The Lab is placed within the Faculty of Business and Social Sciences, one of four faculties at Aarhus University. In order to function, the lab follows a certain set of regulations that decided by two advisory boards.

The first board, the Scientific Advisory Board of Aarhus University is comprised of: Per Baltzer Overgaard(Vice-dean), Alexander Koch(Professor), Andreas Roepstorff(Professor with special responsibilities), Joachim Scholderer (Professor), Michael Bang Petersen. The second board, the Ethics Advisory Group is comprised of: Emma von Essen(Assisntant professor), Joshua Charles Skewes(Associate professor), Julia Nafziger(Associate professor), Martin Bagger(Martin Bagger).

Aarhus BSS represents the main source of funding for Cognition and Behavior Lab. On occasion, lab management have the possibility of applying for grants given by different organizations, with the intent of further developing laboratory facilities. Such grants may be awarded for purchase of new equipment or for development of other systems and IT solutions that will facilitate the conduction of experiments by researchers.

The purpose of the laboratory is to provide a proper work space for researchers of Aarhus University and affiliates where experiments can be conducted in a ethical and scientifically valid setting. For this , the Lab has set it's home page to provide the necessary information and certifications such as: ethics certifications and approvals, prestudy forms and resources guides. The Lab also has a payment policy to help ease the process of reimbursing the participants for their cooperation in the experiment.

For it's participant pool , the laboratory makes the application process for experiments very easy through the use of a user management system and a mobile app. It also put at the disposal of the possible participant several forms informing them of the rights that they have as participants of the Aarhus University conducted experiments.

The Laboratory facilities currently comprise of two computer labs, in which approximately 30 computer units are at the disposal of researchers to run interactive social and cognitive experiments and gather data from subjects; three smaller labs where individual participants can be processed; meeting rooms, a sample preparation area and also a flexible laboratory where more complicated setups can be used.

The laboratory has also a number of cognitive data gathering tools at their disposal such as eye tracking technology andpsychopatologic, brain and nervous impulse receptors, and BIOPAC systems. More information about the lab facilities and equipment can be found on their homepage. (Dan Mønster (2014))

Cognition and Behavior lab have also been building a participants pool for researchers to recruit from as long as the experiment follows all the ethical guidelines that the participants have agreed to when signing up for the participant pool. Thus , every citizen has a chance to contribute to research by being a experiment participant and has the possibility of getting reimbursed for that.All researchers at Aarhus University have the possibility to conduct their experiments at the Cognition and Behavior Lab.

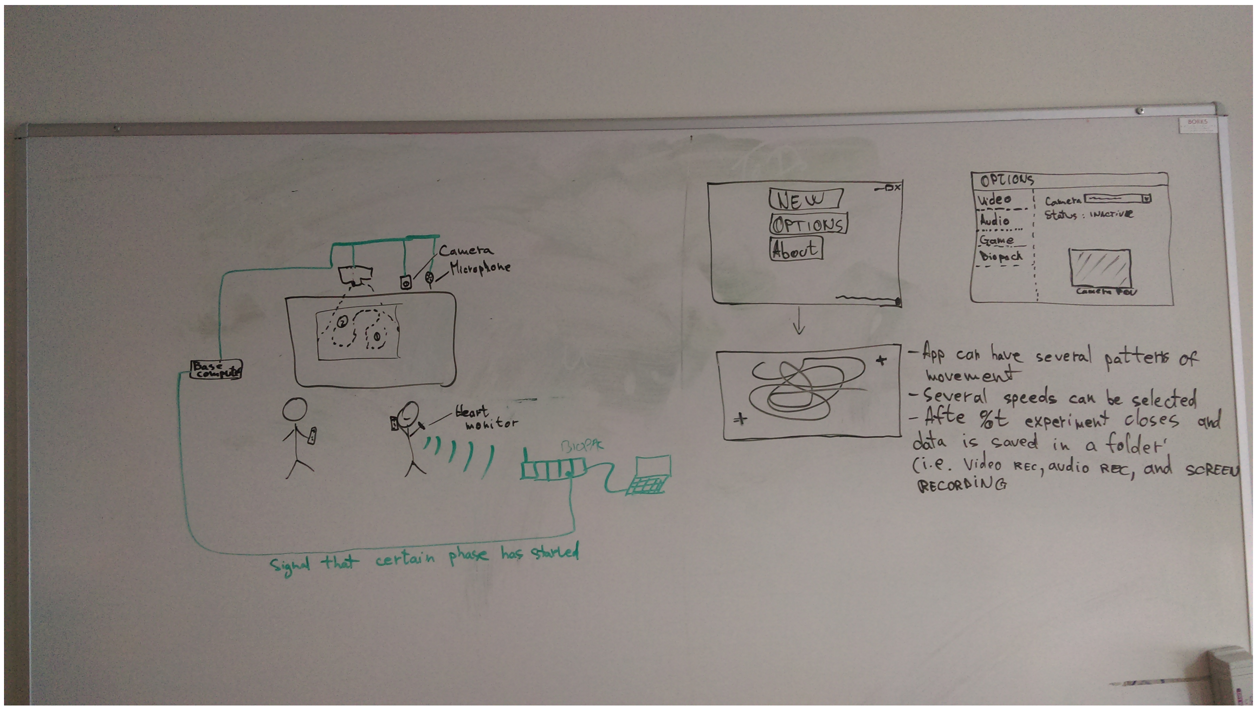
4 Project development and choice of methodology

4.1 Project set-up

As mentioned previously in the Problem Statement on page 3, for the main project I have continued to develop and improve a project previously started in the internship. I have mentioned on a general note what the internship project features were and what my intended direction would be in their development. In this section of the report I will go more into detail about the choices I have made before starting the development process.

4.1.1 Hardware setup

In the planing of this project, the hardware setup was not something that I did not have a deciding factor over. The reason for that being that the setup was aimed for gathering specific research data that the holders needed. For the purpose of clarification, I drew a diagram of how I understood the setting in which the software would be used will look like. I then presented this diagram to the project owner, Dan, to compare our visions and clear up any misunderstandings.


Camera
Microphone
Base computer
Heart monitor
BIOPAC
Signal that certain phase has started
NEW
OPTIONS
About
OPTIONS
Video
Audio
Game
Biopack
Camera
Status : INACTIVE
Camera FOV
- App can have several patterns of movement
- Several speeds can be selected
- Afte %t experiment closes and data is saved in a folder'
(i.e. Video REC, audio REC, and SCREEN RECORDING

This diagram depicts how the software is intended to be used. The two actors(experiment participants), will using a set of controllers(Wii remotes)[1], to interact with the screen which will be projected on a white board. On the screen ,C.O.B.E.S. will give the participants the task to read a text either,synchronously or asynchronously. In both of these modes a reading speed will be imposed by the software through the use of a pointer that will scroll along the text. They will also be asked to read the text synchronously and left to go at a pace which the two participants will decide.

The second task that the participants will perform is perform a cognitive task, where they will be tasked with following a circle on the screen using. Each of them will use a controller to maneuver a point on the screen that will have to stay in the circle that is maneuvered by the software.

While the participants are performing these tasks, microphones will record their speech, while the BIOPAC[2] will record the heart rate and pulse signal from participants and the software will record the IR coordinates[3] from the Wii remotes. Although not shown in the diagram, the experiment conductor will be the one in control of the tasks that the participants will be performing and the order in which they will perform them.

[1]Wii remote- The Wii Remote, also known colloquially as the Wiimote, is the primary controller for Nintendo's Wii console. A main feature of the Wii Remote is its motion sensing capability, which allows the user to interact with and manipulate items on screen via gesture recognition and pointing through the use of accelerometer and optical sensor technology.

[2]The BIOPAC system is a hardware solution that enables researchers to record vital signs such as heart rate and pulse, nervous impulses, brain waves etc. from experiment participants.

[3]Infrared(IR) coordinates, are the information transmitted by the Wii remote from the use of its accelerometer and optical sensing technology.

4.1.2 Language and Environment

For the project, I have decided to keep the development in the C# language using Visual Studio 2015 a decision which was made in the initial phases of the internship period. This proved to be a good call as it allowed me to continue developing without slowing down the process by learning a completely new language .

The projects initial intended language was suggested to be Python by the stakeholder Dan Mønster. He considered Python to be a good choice because it was more favorable for fast implementation of the solution.
 , that required time to be invested in learning the language while also developing the product. That would not have been the best idea as the complexity of the project did not allow for any spare time that could be dedicated in that direction.

In C# however, I was much more comfortable to work in. The experience I have acquired in previous projects meant that time could be spent efficiently of development and with as little drawbacks as possible when encountering new issues. The fact the C# developer community is a fairly active one meant that I would also be able to find support or advice from my peers if anything would come to a standstill.

The deciding factor however was the availability of libraries developed to support the use Wii remote HID[4] devices in software not intended for the use the Wii console. While many such libraries existed for various programming languages a particular WiimoteLib v1.7 proved to be well developed and with no existing bugs.The library however was developed specifically for C#. Since the use of Wii remotes was a must for this project, this influenced the decision to use C# and Visual Studio over Python.

4.1.3 External Libraries

For this project, several features requested by the researchers had no means of being implemented in the standard .Net framework. Because of this, some time was invested in researching other external libraries that would be appropriate to use in the project. For each of these external libraries, the criteria which influenced the final decision of using the library or not varied.

Wii remote library: The first features that required research for an external library were the Wii remote controller features. The library that I was going to use had a big impact on the outcome of the project. Wii remotes are not a very common feature in computer programs. For this reason, I researched the library for the Wii remotes before I researched which programming language to use.
 The reason for this approach, library before programming language may seem counter intuitive at first, but is more beneficial if we take into account the context of the matter.

[4]HID- Human interface devices are considered any piece of hardware that allows a user to interact with a software. These include but are not limited to keyboards, mouses , and other controllers.

Because the Wii remote was developed as a controller for a video game console (4.1.3 HID devices) finding a library which was properly maintained and developed without any bugs and with sufficient documentation proved to be a challenge. After a two day spike in which I researched several libraries, the one that seemed to fill all the criteria listed above proved to be the WiimoteLib v1.7 developed by Brian Peek for Microsoft's Coding4Fun website Brian Peek (2009). After the developer community showed big interest in this library the project went on to become a standalone library for implementing the Wiimote HID device into c# software. One of the most preeminent advocates of this device and it's innovative potential is Johnny Lee. Johnny has developed several applications to highlight the potential of the Wii remote. He has also hosted a Ted talk showing the versatility of this device (Johnny Lee (2008)).

Other libraries either used a different native platform and required changes to work on Windows OS as is the case of Python HTDP for Linux. The Python HTDP driver is a tool for tracking for tracking IR points in 3D using the Wiimote and was developed in Google's summer of code 2008. Other libraries had, according to reviews and community discussions, various bugs that were not solved in time. Gradually the interest for these Input interpreting libraries faded with time and as of 2012 the WiimoteLib v1.7 appeared to be the library that allowed me to use all the Wii remote's potential without having to learn a new language.

During my internship, before starting the main project, I used time to experiment with Wiimotelib v1.7. However ,the final decision to keep developing with this library was made in the beginning of this project. After the final decision to keep the current option, major architectural changes occurred in the software(4.4 Project Development).

Audio recording library: For the audio recording feature I used the experience from my internship and took into account aspects that in the future of development would be influenced by the decision I was about to make. Having this in mind, when I made my decision on which audio library to use, I took into account the available learning material available- the more material the easier to use the library; the popularity of the library among developers- the most popular library was bound to be the one with the most possible help when it came to solving unforeseen issues; and the maintenance status of the library.

The first result that I found when I started looking into these libraries was also the one I ended up using. The NAudio library (Mark Heath (2015)), a open source audio and midi library had all the sound processing capabilities that the features needed. The library started being developed in 2002 and has kept on growing through the years implementing more features. It is actively being developed and new features are implemented while the already existing ones were extensively tested. It allowed me to record, process and save sound from various input sources and it had a relatively large user base where I could find answers to any problem I may encounter. All my requirements were met with this first library, which made the decision of using it fairly simple.

Other libraries: For the project, the client requested to have an option of choosing different colours for the Graphical Stimuli feature. Because the .Net WPF framework did not have a UI Element that allowed picking various colors by the user, I imported the library called Wpf Extended Toolkit ((2015)), a stable 3rd party library that features numerous UI Elements fully stable and in continuous development. This was a fast and effective implementation as it took less than a day to research and implement the library fully.

The BIOPAC communication feature also required the importing of several .dll functions from the inpout32.dll in the system. This did not prove to be difficult as there are many resources available both from Microsoft's documentation as well as user created guides for the uninitiated.

4.1.4 HID devices

Wii remote: The main focus of C.O.B.E.S user experience is centered around the use of the Wii remote. As stated in the previous section, the development for Input interpreter libraries for this device experienced a high interest in the period of 2008- 2009 that gave a good head start for the development of several applications that use the device.

The Wii remote was developed as a controller for the Wii console[5] released in 2006 by Nintendo. Nintendo's intention was to introduce a more interactive way of playing video games. For this purpose they developed the controller called Wiimote, a device that which uses accelerometer data and infra-red tracking as an input method from the user to the console, apart from the classical button input. The remote also features a speaker and a vibration motor, which allow the remote to send feedback to the users. The remote is also capable of taking attachments such as the classic Nunchuk attachment from Nintendo which has an accelerometer built in that allows detection of various hand gestures.

C.O.B.E.S. uses only the Wii remote with it's infra red tracking and accelerometer. C.O.B.E.S. also informs the users of the priority of each remote by the use of the blue LED's at the bottom of each remove.

[5]The Wii is a home video game console released by Nintendo on November 19, 2006. As a seventh-generation console, the Wii competes with Microsoft's Xbox 360 and Sony's PlayStation 3. Nintendo states that its console targets a broader demographic than that of the two others.

Microphones: For the audio recording feature C.O.B.E.S. is capable of identifying any audio input type such as 2.5mm jacks or USB audio input. For it's intended use C.O.B.E.S. will use a QUAD-CAPTURE external audio board, which has two dual channel inputs. The voice of our users will be recorded with two microphones, each of the microphones having it's own audio channel. The audio information is saved then in a file inside the project folder.

BIOPAC: The BIOPAC is a completely separated third party device. For this project, the software will only interact with the BIOPAC hardware through the use of a parallel port. C.O.B.E.S. will then send signals through the parallel port which are then recorded by the BIOPAC along with other information that it gathers from the experiment participants. The BIOPAC will record body signals from the participants and will also record the signals it receives from the C.O.B.E.S. This has the role of synchronizing the two data sets, the data recorded by the BIOPAC and the data recorded by C.O.B.E.S. through it's stimuli.

Before C.O.B.E.S. will start recording, the BIOPAC will already be started and will be actively recording information from the participants . Once C.O.B.E.S. starts the experiment, it will send a signal to BIOPAC that will be recorded alongside with the vital signs. C.O.B.E.S. then proceeds to record the audio data, accelerator data, and IR data. Together all these data sets will be used for the experiment study.

4.1.5 Backups

For the backup of my project I have used GitHub as repository. I found it convenient that Visual Studio 2015 had a built in feature for linking projects to repositories and did not have to install any third party software apart from the GitHub add-on featured in the Visual Studio application store. This also made it easy to clone the repository into other computers as long as they had Visual Studio 2015 installed along with the GitHub add-on.

4.1.6 Management Tools and API's

For the time management of this project, I decided to use for the first time the WakaTime API (`https://wakatime.com/`). This API supports several IDE's and takes information about your working session times and quantifies your effective work time while also offering an overall view over the week of your daily session time. WakaTime uses the same link to all the IDE's you use. So for example if I were to have three IDE's in which I am actively working in the day, the logged session hours from those three IDE's combined will show in my daily work sheet, and each individual language I have worked in will have it's own time percentage out of the pie-char that represents a full day. The decision to use WakaTime as a time management API proved to be very beneficial as it allowed me to asses my effective work time and to adjust sprint estimates accordingly for maximum precision estimates.(4.3 Project planning)

4.2 Choice of methodology

4.2.1 Work environment and context

For the choosing of the methodology, I did not take an approach to compare several methodologies. Instead, I analyze my whole situation and chose the work method that seemed best for managing this situation. I took as criteria the team size, available time, availability of project owners and meetings with the stakeholders and how often did the features change through out the development life cycle.

In this project I would be the only developer, manager and designer of C.O.B.E.S. In the office, two other interns were present but they each worked on a project with the same situation as mine. Thus, all the development and designing decisions were left for me to asses and take. We were available for each other at any time for feedback, or advice when we had a problem that seemed to now be advancing as expected.

The start date of the project was set from 5th October and had the deadline set for 16th December. In this time I had to develop C.O.B.E.S. to it's full specifications and perform appropriate testing for quality assurance all in a time span of 10 Weeks.

As project owner Dan Monster was present in the office three days per week. Each Wednesday and Friday he would be out of office to teach at the University classes. Three days a week in which the project owner was always available for discussions and clarifications was a very favourable situation as it allowed for agile work.

During our internship period, the other two interns, Lab Manager and Assistant Lab Manager and I met to discuss our work of the previous week and plan for future progress. The two other interns and I would also have discuss activities every morning while making coffee. In this meeting we would again talk about what we plan to do for the day and what we think would be the difficult part. These meeting had a good team building effect as it made us realize we were all working for a common goal, as well as make us responsible to the other colleagues by informing each other about our intentions for the day. In the case we slacked off we could be held accountable by our peers for not fulfilling our duties. Happily, no such situation happened in the internship or in the main project period.

4.2.2 Going agile and the principles behind it

Using my previous development experience from other projects as well as making use of the fact that I was alone in this project, I decided to work in an agile manner while using parts of Scrum to create a working manner which I am comfortable in working with. My methodology was centered around key principles of the Agile methodologies manifesto such as:

- Customer satisfaction by early and continuous delivery of valuable software
- Working software is delivered frequently, in my case weeks
- Close ,daily cooperation between developer(me) and project owner(Dan Monster)
- Project build with the help of motivated individuals with high level of trust
- Working software is the principal measure of progress
- Continuous attention to technical excellence and good design
- Simplicity — the art of maximizing the amount of work not done — is essential

These principles have guided my approach towards this project, helping me to keep in mind and balance customer satisfaction with practical project progress and attention to technical excellence and good design.

4.2.3 Stop Scrum-ing around

Scrum is a leading agile development methodology for dealing with projects that have a complex innovative scope of work (Schwalbe (2012)). It "is an enhancement of the commonly used iterative and incremental object-oriented development cycle" and involves implementing a small number of customer requirements in sprint cycles of two to four weeks (Schwaber (1995)). Scrum does not have a fixed set of engineering procedures or guidelines. It also does not impose specific managerial processes so that it avoids chaos through it's various stages.(Yousra Abdo Hardb , Cherie Noteboom , Surendra Sarnikar (2015)). These advantages of the Scrum methodology were ideal for the project characteristics imposed by the client and work environment.

The absence of guidelines to impose a specific working ethics allowed me more flexibility when it came to research for the requested features. Using Wii remotes as HID devices was not something I had previous experience with. It was also one of the main features requested by the client that had a huge impact on the success rating of the project. Because of this, a lot of stress has been put on the proper implementation of the feature and the user experience quality of it.

Having the client and project owner, Dan Monster, in close proximity to the workspace meant a good opportunity to ensure that no communication problems existed between the client requests and the developer perception of those requests. To achieve this I planned on making extensive use of face-to-face communication in favour of written communication. In the beginning of the project, a document describing the expected software was handed out to me. But from the beginning of the project until the end some features either had to change or my understanding of how they were t to be implemented changed. More on this subject can be found in section 5.1 Reflections on chosen methodology. In the life cycle of C.O.B.E.S. I came to the conclusion that in agile methodology, and Scrum specifically, the project owner becomes part of the team as he has as active role in steering the project development to match the developer vision with that of the clients.

For keeping track of the progress made, the office made use of several Scrum specifics. First of all, the office had a weekly Scrum meeting for briefing purposes. All members were to participate in these meetings to discuss the latest developments in our projects as well as discuss the future plans and get advice. The 3 of us interns also had a daily Scrum where we discussed what we were planning to do for the day. This made us accountable for each other in the eventuality that we do not keep to our plans. As a main measure of success rate, working software and client approval was used. In this way, we had a good idea of how far we came. The feature was only considered a success once its full development was going to be done. That meant finished design and architecture as well as performing under expected parameters from the client.

By using all the above mentioned principles I was expecting to keep a steady work flow of research and development. The use of concrete achievements i.e. working software as a measure of success allowed for a realistic estimate of progress.

4.2.4 Alternative methodology

The other methodology that I had in mind to use was the Extreme Programming(XP) methodology. Extreme Programming is the other largely known agile programming methodology. Many aspects of extreme programming are shared with Scrum. Both methods are based around the same principles. They both work in a dynamic manner that will allow for easy and fast adaptation to ever changing requirements. They both focus on a good relation and communication with the client to prevent bad communication as much as possible and they are both based on small teams.

However, despite all these similarities, I do not believe XP methodology to be a good fit for this project. The XP methodology makes use of some processes that might not help this project and that might even be detrimental. The XP iterations, equivalent to the Scrum sprints, are supposed to be longer. There are no necessary daily scrums and certain engineering methods are encouraged such as pair programming.

Seeing as for this project I was a one-man team I did not see the benefit of having longer iterations or using pair programming. However I found useful the idea of having daily Scrums and weekly Scrum meetings with my peers. The longer iterations meant also that agile development factor would be lower.For a project to be successful in this circumstances would mean that previously earned knowledge of the software engineering development would have to be know. Otherwise iterations might prove to be unsuccessful because of the lack of knowledge.

In Scrum, the shorter sprints meant more opportunities to adapt the process to new problems that might arose. It could be possible that some delay might be caused because I will lack knowledge of a certain framework. In Scrum that would not be a very big problem as the sprints are two weeks long and can easily take that into account when planning the next sprint with the team.

If however I would work in XP iterations that would last four weeks, I would have to wait until the next planning meeting to assess my progress with the team and adapt the process then. This could mean a potentially big setback in the end.

4.3 Project planing

For the planning of the project, I decided to take a feature driven approach. The main reason for this decision was that it was a very palpable way of measuring progress. As mentioned in the previous chapter of choosing the methodology. This had the added benefit of limiting chaos in the workspace by not imposing a specific management tool.

In order to properly estimate what exactly can be accomplished within a single sprint, I took into account the previous experience from the internship period as well as records of my work hours taken with the WakaTime API. The WakaTime API and previous experience allowed to have a realistic overall view of what was achievable in a given time period. Given the fact that the project lasted only 10 weeks, there was also concern expressed by the project owner regarding the meeting of deadlines.

4.3.1 Sprints

Being a one-man team, and having several features to implement by the deadline, the sprints had the total length of two weeks each. Each sprint started on a Monday and ended the following week on Friday. Each of the sprints also had a burn-down chart with a backlog established in the first day after the weekly scrum meeting. The second scrum meeting of the sprint came at the halfway point in the second Monday of the two week period when the project owner was going to be informed of the progress of the current sprint.

The ten week period was divided into five total sprints, each of them focusing on one of the core features of the project, and in some cases several smaller tasks. Each of these sprints also came with a backlog of the Feature to be implemented, divided into the expected tasks that needed to be performed. The estimate of the expected tasks came from the experience obtained during the internship, in particular from experimenting with the external libraries.

Planed sprints:

- First Sprint: **feature:**Wii remote; **duration:**2 weeks(5 Oct-18 Oct)
- Second Sprint: **feature:**Prompter window; **duration:**2 weeks(19 Oct-2 Nov)
- Third Sprint: **feature:**Audio recording; **duration:**2 weeks(2 Nov-16 Nob)
- Forth Sprint: **feature:Miscellaneous user stories; duration:**2 weeks16 Nov-30 Nov
- Fifth Sprint: **feature:Sequence execution,Data recording; duration:**2 weeks1 Dec - 14 Dec

Every sprint consisted of a research, design, and development for each user story planned for development. In the research phase, I would look into different options of implementing the desired feature. After finding a satisfactory solution, the following step take was designing or redesigning the architecture, depending on previous features. Finally, after the design, came the development part of the sprint. Working in an agile manner, these steps were repeated several times in a single sprint, depending on learning necessities, changes in client requirements, or in some cases clarification of miscommunication between project owner and developer. A buffering period was in the sprint was also taken into account to compensate for eventual delays that might happen. I will describe each sprint in detail in section 4.5 Description of the process.

4.3.2 Acceptance meetings

To keep all the project stakeholders up to date acceptance meetings were planned to be held. The date was not pre-set, but rather it was decided to conduct a meeting only when significant progress had been made and deemed worthy of presenting(i.e. content-driven meetings). Another reason was also that the other stake-holders were not always available for discussion except for the project owner. It could prove more difficult for them to be present at those meetings if other tasks interfered with the plans.

Apart from the schedule of the project stakeholders, another factor that needed to be taken into account when planning an acceptance meeting was the availability of facilities. For the intended use of C.O.B.E.S. certain requirements needed to be satisfied such as enough space for the participants and privacy for the duration of the experiment. Such conditions could be offered only by a limited number of rooms, which were also used by other researchers at the laboratory.

The first acceptance meeting was held on 13th November with a follow up planned in the last days of the project development. We also then expect to conduct pilot testing, to find any last adjustments that can be made to the entire set-up, not just the software.

4.4 Project development

As previously mentioned, when I started the main project period on the date of 5th October, I already had an existing project which was started in the internship period. That previous version however, was in no functioning condition. Some features were implemented to a certain degree but with many faults that made them unusable.

Wiiremote functionality The Wii remote control had several problems to start with. It was possible to connect the remote to the computer through Bluetooth and take the data from it with C.O.B.E.S. but there were problems in managing that data. The IR Coordinates provided by the remote were going to be used to control a point on the screen with each remote. Passing that information on to the GUI however turned out to be a big challenge.

In the first model of architecture a Wii remote handler would be created for each of the physical controllers. That handler would create and handle the data received from the device and pas it on to the GUI through the service class. This method however proved to be very inefficient as the main thread of the program had trouble keeping up with all of the information that it had to display. Pinpointing this problem however proved another challenging task as there could have multiple reasons for why the graphic feedback was the way it was. Given that I was working with a physical device, some engineering issues might have been to blame such as the remote not detecting the IR[6] light source properly to get a position on it.

Microphones functionality The microphone functionality made it possible to save audio recordings into files inside the project file. The software offered the possibility of having a maximum of two audio inputs to save the file from, but there was no fail-safe to prevent users from selecting the same audio input in both of the available audio slots. This could very easily lead to situations where there would be two audio files with the same content. On other occasions there were also audio files that although had been recorded with no errors, they would either have no data in them or they would show up as not being properly closed or still being used by a program when I tried to open them. From the microphone feature though, I found that the architecture model that I implemented was very intuitive and easy to work with.

After implementing this architecture and seeing how easy it was to iterate over it, I decided to implement it for the other devices that needed to be performed by the software.

Stimuli windows features Before the start of the main project period, the stimuli windows that were already present in the software, were prototypes of what needed to be implemented. Each of the two windows had one running mode implemented, which was the synchronous running mode. In the movement task, the users had to both follow a circle using the controllers while in the reading tasks they had to read a text together. In the current state that the stimuli windows

The following phases that needed to be done were implementing the asynchronous and self-paced running modes for both the movement and reading tasks.

[6]IR-LED stands for Infrared Luminous Electric Diode

Recording feature The recording feature for information gathering was almost finished before the main project period started. Little adjustments needed to be made such as the frequency of the data recording and the format in which the client wanted it saved but that did not come up until after our first acceptance test meeting. The feature consisted of a single timer that would be started each time a movement task stimuli window was being ran. The timer would then send information to a container class where data would be taken and stored in a file belonging to each instance of a ***StimuliWindow*** class.

GUI The graphical user interface of C.O.B.E.S. evolved along with the software features. Throughout the whole project, there has been a direct positive correlation between the complexity of the GUI and how many of the features were implemented properly. So far before the main project period , the GUI had a menu for connecting the Wiimotes to C.O.B.E.S. as well as giving visual feedback to the user that the connection was successful. The feedback came as real time data displayed on the screen of the information that the remote was sending to C.O.B.E.S. This information came in the form of ***float*** values for the three accelerometers each of them being attributed to one of the three dimensional axes X,Y or Z. The GUI also displayed to values that the remote was detecting for each of the 4 IR-LED's emitters. This information came in the form of float value pairs. Each pair belonged to one of the LED'S and was consisting of a value of the X and Y corresponding to a two dimensional Cartesian plane. The remote had a possibility to give information concerning the depth of field to which it was detecting the light sources, however that was not a requirement and would not have brought any extra benefit for our purposes.

4.4.1 Initial Architecture

After having experienced first hand what problems can occur in the project because of poor architectural decisions and bad engineering decisions, the first step I took in the project development was to reiterate over the architecture model. The first architecture type that I had was as following:

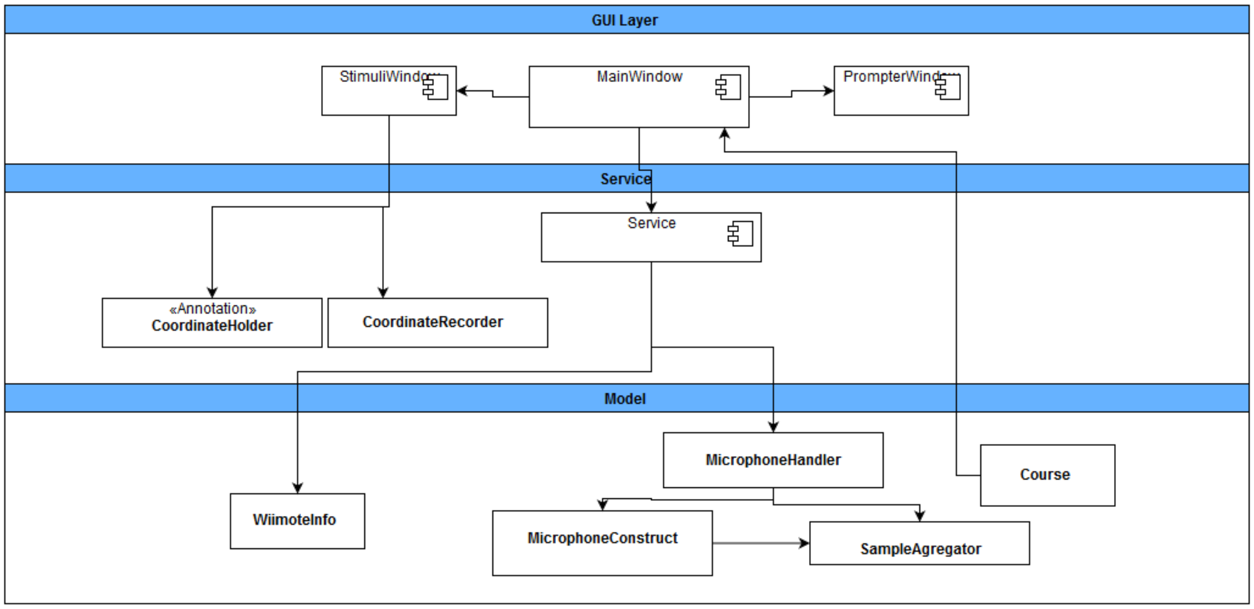


Figure 1: Arhchitecture 1.0

From the diagram we can see that there was no particular idea to which I adhered to when planning out the software. My base ideas was to use a three layered architecture to have separation by tasks.

The ***Model*** layer consisted of the base classes that were used to manage the different devices and and features. In the ***Model*** layer the ***WiimoteInfo*** class was used to handle the Wiimote presence in the software. Here is where the connection to the devices was established and processed through the whole application run time. This was the same class setting that the WiimoteLib used in the application that came along with the library. The application called test.exe was made to highlight the basic capabilities of that library. However this model was made for the use of a WinForms based application. Although in this demo software the use of multiple Wiimotes seemed fairly easy to do, this was not the case for a WPF application that I was making.

The ***Courses*** class present in the model is one of the classes that remained largely unchanged from start to finish. The class was responsible with providing the coordinates that the stimuli window was going to use in it's task, which the participants were to follow in the movement task, was going to use as it's trajectory throughout the whole task. Each time a movement stimuli window would be created it would have a ***Course*** object that would provide it with coordinates.

The ***MicrophoneHandler, MicrophoneConstruct*** and ***MicrophoneSampelAggregator*** classes fulfilled a specific role in helping the management of multiple audio inputs.

The ***MicrophoneConstruct*** class was the base class. Here every audio input connected to the computer, regardless of the input type (USB, 2,5 mm jack) would be retrieved from the operating systems device list and have it's address saved into memory. The class also made it possible to create recordings and save them, all of this with the help of the NAudio library.

The ***MicrophoneHandler*** class was the class that handled every MicrophoneConstruct object that was created. Here the software would keep a list of all the devices that were currently connected to the computer and a list of the two devices that were currently being used for recording purposes.

The ***MicrophoneSampleAgregator*** class is where the information from the microphone is processed before being recorded and sent to the GUI to offer the user a graphical feedback of his input. This class has the main purpose to process the number of audio samples and prevent excessive use of resources for recording purposes. In the software the recording frequency is set to be 44100 Hz however we do not need that amount of detail into our recording. Filtering the sample number allows us to maintain a good quality of audio recording while also keeping the available resources.

For recording purposes the ***CoordinateRecorder*** was used along with the ***Coordinate-Holder***. The ***CoordinanteHolder*** would receive information from the ***StimulyWindow*** periodically. At the same time, the ***CoordinateRecorder*** would access that information, format it, and store it in a .txt file that was made the moment the command to start recording was received. The reason why I found the ***CoordinateHolder*** necessary was due to how the threading in this architecture worked. The ***StimulyWindow*** had, in its own graphical thread, a ***Timer*** object that would send the information to be recorded. On the other end, in the ***CoordinateRecorder*** , a different ***Timer*** would format and the retrieved information and save in the file. In order to prevent threading problems such as conflicts over which thread would have access to the data, I found it easy to make a tampon zone where one thread would be allowed to write while the other would only be allowed to read data.

In the end several modifications had to be made both to the architecture model of the feature and to the engineering perspective of the code. These changes are going to be discussed in the following section with more detail regarding to what issues they addressed as well as how they were implemented.

4.4.2 Wii remote feature

The Wii remote controller was implemented initially with the use of a single class, ***Wi-imoteInfo*** . Through this class connecting and receiving the information from the device was made possible.

When instantiating the class, a connection is made to the owner of the object and a new collection of Wii remotes is created and retrieved from the the operating system.

```
public WiimoteInfo(Service observer)
        {
            this.mWiimotes = new WiimoteCollection();
            this.observer = observer;
        }
```

After the constructor is called and the collection of Wii remotes is retrieved, connecting to them is one by the ***Connect*** method, which uses as parameter the index of the remote we want to connect to. The remotes are ordered in the way they are connected to the Bluetooth devices in the operating system.

```
public void Connect(int i)
        {
            try {
                mWiimotes[i].Connect();
                mWiimotes[i].SetReportType(InputReport.IRAccel, true);
                observer.SendMessage(i+1 , "Wii remote connected ;");
                if (i == 0)
                    mWiimotes[i].SetLEDs(true, false, false, false);
                if (i == 1)
                    mWiimotes[i].SetLEDs(false, true, false, false);
                mWiimotes[i].WiimoteChanged += wm_WiimoteChanged;
            }
            catch(Exception ex)
            {
                Console.WriteLine(ex.ToString());
                throw;
            }
        }
```

After the connection was made, the Wii remote is told what type of information we want to receive. In C.O.B.E.S. we only needed to use the accelerator and infrared sensor information. Then, a message is sent to the observer and the remote LED is light to show which one is the first remote connected and which is the second. This helps to avoid confusion during the use of the feature.

Whenever the status of the remote is changed, because it was moved and the accelerator values have changed or because a button was pressed, the ***Wiimote*** object calls the ***WiimoteChanged*** even handler which will process the new data in whatever way I needed it to. In this case, the ***wm_ WiimoteChanged*** method calls and awaits for the ***UpdateWiimoteChanged*** to process the data.

```
private async Task UpdateWiimoteChanged(WiimoteChangedEventArgs args
, object sender)
        {
            WiimoteState wiimoteState = args.WiimoteState;
            Accelerometer[0] = wiimoteState.AccelState.Values.X;
            Accelerometer[1] = wiimoteState.AccelState.Values.Y;
            Accelerometer[2] = wiimoteState.AccelState.Values.Z;

            this.UpdateIR(wiimoteState.IRState.IRSensors[0], 0);
            this.UpdateIR(wiimoteState.IRState.IRSensors[1], 1);
            this.UpdateIR(wiimoteState.IRState.IRSensors[2], 2);
            this.UpdateIR(wiimoteState.IRState.IRSensors[3], 3);

            if(wiimoteState.IRState.IRSensors[0].Found &&
            wiimoteState.IRState.IRSensors[1].Found)
            {
                this.UpdateIR(wiimoteState.IRState.RawMidpoint, 4);
            }
            if (wiimoteState.IRState.IRSensors[0].Found &&
            wiimoteState.IRState.IRSensors[1].Found)
            {
                Console.WriteLine("Found 2 sensors");
            }
            this.battery = (wiimoteState.Battery > 200f ? 200 :
            (int)wiimoteState.Battery);
            await observer.informMainWindow(this, (Wiimote)sender );
        }
```

The information would be processed with the ***UpdateIR*** method to make sure that the values are within the scope that they are expected to be and not cause any exceptions in the runtime.

After this, the information was sent to the ***Service*** through the "await observer.informMainWindow(this, (Wiimote)sender " which then passed along the information to the MainWindow for display.

In order to display a feedback of the remote interaction in the computer, the GUI used had several ***Ellipse*** UI elements which were then manipulated using the information received from the ***WiimoteInfo*** .

```
public void UpdateWM1Labels()
{
 Action action = () =>
            {



                WM1Status_Lbl.Content = wm.WiimoteState.ButtonState.A.ToString();
                WM1_Accel_X.Content = wm.WiimoteState.AccelState.Values.X.ToString();
                WM1_Accel_Y.Content = wm.WiimoteState.AccelState.Values.Y.ToString();
                WM1_Accel_Z.Content = wm.WiimoteState.AccelState.Values.Z.ToString();
                WM1_IR1.Content = String.Concat(
                "; X = ",wm.WiimoteState.IRState.IRSensors[0].RawPosition.X / 10,
                "; Y = ",wm.WiimoteState.IRState.IRSensors[0].RawPosition.Y / 10,
                "; Size = ",wm.WiimoteState.IRState.IRSensors[0].Size + 1);

               if (wm.WiimoteState.IRState.IRSensors[0].Found)
                {
                    IRSensor11.Visibility = System.Windows.Visibility.Visible;
                    Canvas.SetRight(IRSensor11,
                    wm.WiimoteState.IRState.IRSensors[0].RawPosition.X/10 * 3 );
                    Canvas.SetTop(IRSensor11,
                    wm.WiimoteState.IRState.IRSensors[0].RawPosition.Y /5 );
                }
                else
                {
                    IRSensor11.Visibility = System.Windows.Visibility.Hidden;
                }

                //..........This whole proccess will be repeated a total of 4 times in
                //when the Dispatcher invokes the action.
        try
            {
                await Dispatcher.BeginInvoke(action);
            }

            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }

        }
```

This lengthy method would then be used every time there was a new ***WiimoteChange*** event so that the Wii remote menu would display feedback about the interaction between the users and the C.O.B.E.S. through the Wii controllers.

After this implementation however, it became obvious that the information flow from the model layer all the way to the GUI layer was not good enough. Our main problem with this approach was that at certain points the graphics would start to freeze up and performed jagged movements instead of having a smooth linear movement as we expected. After some investigation into the issue, time in which I looked into the variables that could affect the information flow, I came to the conclusion that the movement problem was duo to a conflict of resources between the two remotes calling the ***Dispatcher*** of the main GUI thread simultaneously. That generated a conflict of interest as there were several hundred calls each second. It became very obvious that handling every ***WiimoteChanged*** event in the same thread and methods was not a good idea.

Initially, I though that a way to fix the problem outlined above, was to create two separate threads in the main GUI thread which would then handle the two data sets coming from the Wii remote controllers. I quickly found however, after looking for answers online, that this was impractical and close to impossible. Each window in the .Net/WPF framework was made to have it's own thread which handles all the UI elements. To make two child threads inside the main thread, with the purpose of handling the data influx from the controllers was not a solution. Displaying that data was done in UI elements belonging to the parent thread. That meant that both of smaller threads should have been to access different data simultaneously in the main thread which was not easy to do, if not impossible.

However, while searching, I came by a design pattern which after reading up more on it, seemed to have the answer to my dilemma. The MVVM pattern, or Model-View-ViewModel, is a design pattern that was developed by Microsoft's architects Ken Cooper and Ted Peters. The aim of this pattern was to take advantage of the features of Windows Presentation Foundation (WPF) (Microsoft's .NET graphics system) and Silverlight (WPF's Internet application derivative) and ultimately simplify the event-driven programming of user interfaces.(Smith (2008))

The MVVM rationale was to make use of the DataBinding functions in WPF(Windows Presentation Foundation) to better facilitate the separation of the view layer development from the rest of the pattern, by eliminating the GUI code from the view layer (Smith (2008)). The MVVM pattern required user experience (UX) developers to use data bindings in the framework mark-up language (e.g. XAML) and link the GUI to the view model, which is then written and maintained by application developers. This role separation allowed the interactive developers to focus on UX rather than business logic.

In my case however, the MVVM came to be only partially implemented due to the lack of time. Based on the MVVM pattern, I designed the whole interaction between GUI layer and Wii remotes completely on data bindings and event-driven architecture. Since the WiimoteLib was already developed to make use of events for communicating with the computer, my part remained to make use of the data binding feature of WPF so that the GUI could automatically respond to the constant changes thanks to event. This ultimately removed any conflict over the resources of the main thread and allowed for a smooth user interface with fast feedback.

After implementing the MVVM pattern, the ***UpdateWM1Label()*** method was replaced by a ViewModel class as that linked the GUI to the model layer. All the attributed that were set in this method were ported to the ViewModel class, and data bindings between them an their respective UI Elements were established.

That however was not the only change that occurred to the Wii remote feature. Before I could say that this requirement was completely finished, I decided that for ensuring a good quality of code and a possibility to extend the program in the future, a change in the model layer was necessary. What was previously the ***WiimoteInfo*** class, I separated into three different classes much like the microphone classes. The initiation of each Wii remote device and all the setting that needed to be effected on the remote were moved to the class that is ***WiimoteHandler*** . The function of processing the data that came from the Wii remote controller was given to the ***WiimoteSampleAggregator*** . The role of passing the coordinates on to the ViewModel layer and the databindings was attributed to the ***WiimoteCoordiate*** class.

Currently, whenever a new Wii remote controller is connected, the ***WiimoteHandler*** adds a new ***Wiimote*** [7] object to the ***WiimoteCollection*** that it manages. After adding the new remote, it creates a new ***WiimoteSampleAgregator*** and a ***WiimoteCoordinate*** that are attributed to that ***Wiimote*** . Each time the device will give WiimoteChanged event, the status of the remote will be sent from the handler to the aggregator class where it will be processed so that we only have the information we need. Once that is done, the aggregator will notify the handler that it has finished processing and the handler will pass the information along to the ***WiimoteCoordinate*** object belonging to our remote instance. Here the information is sent out to the ViewModel layer through the use of event notifiers.

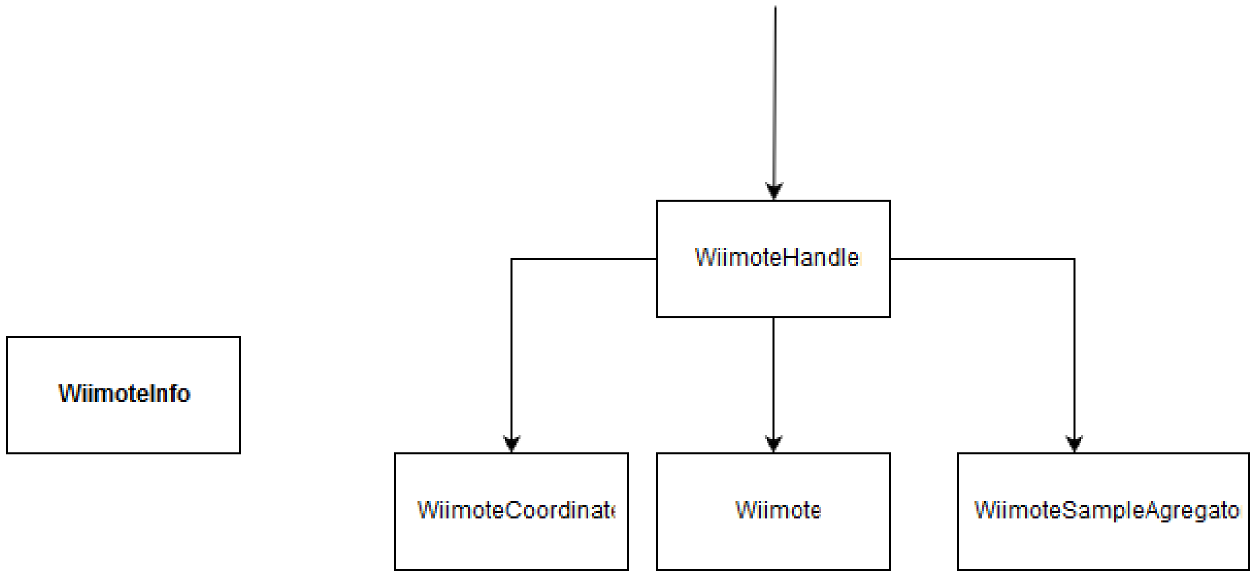


Figure 2: Before and after class role reassignment

4.4.3 Microphones and the recording feature

The audio recording feature, has maintained it's base architecture throughout the entire period of the project. Some small changes had been done or things added in it's classes in so that the user experience could be improved, changes that will be explained later. The model of the feature has remained the same as described previously in section 4.4.1 Initial architecture and is very similar to the one that the Wiiremote feature uses. This is because, this model was first implemented in the audio handling and was determined to make use of it further seeing as it was a very intuitive way of architecture design. There are however, some differences between the two implementations.

The ***MicrophoneHandler*** class, is responsible with handling all the audio devices that are present in the operating system memory. Whenever the software is started, a handler object for the microphones is created by the service class. The handler will then check to see

[7]Wiimote and WiimoteCollection objects are a classe defined in the WiimoteLib. The C.O.B.E.S. project is only making use of them in it's features.

that the folder where audio recordings are expected to be saved is actually present in the assembly location. When the main window has finished loading, it will request the service class to provide a list with all the available audio input devices from the ***MicrophoneHandler*** class. The handler then uses the ***NAudio*** library and its WaveIn [8] to find all the devices and retrieve their information such as device number in the operating system memory, device name etc. and saves them in the handler class before providing the list of active devices to the service which then passes it to the GUI.

```
public List<MicrophoneConstruct> MicrophoneList()
{
    int WaveInDevices = WaveIn.DeviceCount;
    microphones = new List<MicrophoneConstruct>();
    for (int WaveInDevice = 0; WaveInDevice < WaveInDevices;
     ↪  WaveInDevice++)
    {
        MicrophoneConstruct deviceInfo = new
         ↪  MicrophoneConstruct(WaveInDevice);
        deviceInfo.aggregator.MaximumCalculated +=
         ↪  MaximimumCalculated;
        microphones.Add(deviceInfo);
    }

    return microphones;
}
```

The "microphones" list is made of ***MicrophoneConstruct*** objects which have the responsibility of encapsulating and setting up each audio input provided by the ***WaveIn*** . The class also attributes a ***SampleAggregator*** to each input so that the audio samples are filtered for memory efficiency purpose as well as analyzed for in order to determine the volume of the input and display it as graphical feedback on the GUI.

[8]The WaveIn object provided by the NAudio library allows access to the devices currently present and active in the operating system.

```
public MicrophoneConstruct(int selectedDevice)
        {
            waveIn = new WaveIn();

            if (WaveIn.DeviceCount > 1)
                waveIn.DeviceNumber = selectedDevice;

            waveIn.WaveFormat = new WaveFormat(44100, 2);

            waveIn.DataAvailable += new
             ↪  EventHandler<WaveInEventArgs>(waveIn_DataAvailable);
            waveIn.RecordingStopped += new
             ↪  EventHandler<StoppedEventArgs>(waveIn_RecordingStoped);

            aggregator = new SampleAggregator();
            aggregator.NotificationCount = 10; // How often we update the
             ↪  volume bar value;
        }
```

Figure 3: Creating a microphone construct in memory

When creating a new device in memory, the device is retrieved from the ***WaveIn*** with the use of the "selectedDevice" integer.The format of the microphone is then set with the "waveIn.WaveFormat = new WaveFormat(44100, 2)" which defines the microphones input as being a 44100 kHz recording with 2 audio channels (left channel and right channel). These specifications are that of a standard stereo recording. Two event methods are then attributed to the selected WaveIn, one to handle the audio samples registered (waveIn.DataAvailable) and the other to handle the recording stopped event. Finally a ***SampleAggregator*** class for each microphone is created. The reason that each microphone has it's own ***SampleAggregator*** is so that multiple callings on the same object are not done and thus avoiding a conflict for resources. The second reason is that each microphone will have different input values and so different volumes that need to be displayed simultaneous on the GUI.

```
public void Add(float value)
            {
                maxValue = Math.Max(maxValue, value);
                minValue = Math.Min(minValue, value);
                count++;
                if (count >= NotificationCount && NotificationCount > 0)
                {
                    if (MaximumCalculated != null)
                    {
                        MaximumCalculated(this, new
                         ↪  MaxSampleEventArgs(minValue, maxValue));
                    }
                    Reset();
                }
            }
```

Whenever new audio samples are sent to the ***SampleAggregator*** of the ***MicrophoneConstruct*** , the add method uses the float value of the sample to process it's maximum and minimum value which are then stored in the ***SampleAggregator*** memory. An event is fired every time the maximum and minimum values are reassigned and the values are then displayed on the GUI in form of a volume bar so that users know when the microphone they have selected is processing the audio input.

When the user needs to select a microphone to use, he is given the option to go to the audio recording section in the GUI. There he will select from a list the microphone that he wants to use. It is easy to identify the microphones as they are being displayed in the list by the names that the operating system identifies them with. When the item is selected in the list, the GUI calls a method in the ***Service*** class, that then calls the listen method in it's ***MicrophoneHandler*** object.

```
  public bool ListenToMicrophone(int selectedDevice , int groupBoxIndex)
        {
            MicrophoneConstruct mic = microphones[selectedDevice];

            mic.Listen();
            activeMicrophones[groupBoxIndex-1] = mic;

            TryGetVolumeControl(selectedDevice , groupBoxIndex);
            return mic.Active;
        }
```

Figure 4:

When a ***MicrophoneConstruct*** object is being listened to, it is added to a list of active microphones that are displayed as the ones in use. The feedback is being displayed on the

GUI by making use of the MVVM pattern and data bindings in the same manner that the Wii remote feature does through the Service class once more.

4.4.4 Stimuli Windows- Movement Tasks

Because the participants need to perform two separate types of task, each of them with three different conditions, the software had two types of windows made in the GUI specifically for these tasks. One window, the ***MovementWindow*** that allows the users to use the Wii remotes and control pointers on the screen with which they will have to perform three different tasks. The second window is the ***ReadingWindow*** class. Here participants will be given a text that they will have to read in one of the different modes.

In the Asyncrhonous mode, user will have to perform a task in a relay manner. A first ellipse, called a ***stimulyEllipse*** , will move along the map and one user has to keep his pointer in that circle while it moves. In that time, the secondary player will move to a position that will be shown through the use of a secondary ellipse called a ***checkpointEllipse*** . Once the ***stimulyEllipse*** reaches the checkpoint, the ***checkPointEllipse*** will move to the following location where now the primary user will have to wait while the secondary user will be the one who will now follow the ***stimulyEllipse*** . This will be repeated until the whole set trajectory will be finished. In order to make it easier for the users to keep track of who has to follow and who has to wait at the checkpoint, the two ellipses will always switch colour so that, the player who has to follow the ellipse will have the same pointer color as that ellipse and the player that has to wait at the checkpoint will have the same pointer color as the ***checkPointEllipse*** .

In the Synchronous mode, users will perform the same task as before only simpler. Here the users will both need to follow the ***stimulyEllipse*** through the whole course and no ***checkPointeEllipse*** will be present on the screen.

In the Self-Paced mode, the users will be tasked to follow along a line and stay as close to each other as possible. The ellipses will play no part in this mode, and in stead the trajectory on which the ellipse would move will be shown. That trajectory is what players will follow along for the duration of the session.

The ***MovementWindow*** had in the beginning only one option for the way the course was going to be played. The window would generate a ellipse in which users would have to follow on the window with the help of the Wii remotes. During the project however, the window had to be developed further to have the possibility of the other types of courses to be played.

Whenever a new ***MovementWindow*** is being started from the ***MainWindow*** , it will be started with several options that the user will set up in the menu. The user has the ability to choose between three different types of courses: the SelfPaced mode, the Asynchronous mode, and the synchronous mode. The asynchronous mode and the Synchronous mode will generate a new ***MovementWindow*** using the same constructor.A delegate method called "generate" will then be used to create the appropriate course mode.

```
public MovementWindow(MainWindow mainWindow , string mode , int complexity
 ↪  , int Speed, System.Windows.Media.Color color1 ,
 ↪  System.Windows.Media.Color color2 , System.Windows.Media.Color color3 )
        {

            InitializeComponent();

            //Setting up Ellipses and pointers
            Pointer1.Stroke = new SolidColorBrush(color1);
            this.color1 = new SolidColorBrush(color1);
            Pointer1.DataContext = model;
            Pointer2.Stroke = new SolidColorBrush(color2);
            this.color2 = new SolidColorBrush(color2);
            StimulyEllipse1.Stroke = this.color1;
            CheckPointEllipse.Stroke = this.color2;
            Pointer2.DataContext = model;
            lineBrush = new SolidColorBrush(color3);
            CourseSpeed = Speed;

            //Queue of animations that will move the ellipse
            this.queue1 = new AnimationQueue(StimulyEllipse1,
             ↪  Canvas.LeftProperty, Canvas.TopProperty);
            this.mainWindow = mainWindow;
            recorder = coordinateRecorder.getInstance(this);
            holder = CoordinateHolder.GetInstance();

            t = new System.Timers.Timer();
            t.Elapsed += new ElapsedEventHandler(SendInfo);
            t.Interval += 100;


            this.SetBinding(Window.WidthProperty, new Binding("RezolutionX")
             ↪  { Source = model, Mode = BindingMode.OneWayToSource });
            this.SetBinding(Window.HeightProperty, new
             ↪  Binding("RezolutionY") { Source = model, Mode =
             ↪  BindingMode.OneWayToSource });

          if (mode == "Asynchronous")
            {
                generate = Asynchronous;
                checkPointTimer = new System.Timers.Timer();
                checkPointTimer.Interval = 1000.0 / CourseSpeed * 50;
                checkPointTimer.Elapsed += new
                 ↪  ElapsedEventHandler(ShowNextCheckPoint);
                queue1.checkPointHandler += new
                 ↪  EventHandler(ShowNextCheckPoint);
                StimulyEllipse1.Visibility = Visibility.Visible;
                CheckPointEllipse.Visibility = Visibility.Visible;
            }
            if (mode == "Synchronous")
            {
                generate = Synchronous;
                StimulyEllipse1.Visibility = Visibility.Visible;
            }

        }
```

Figure 5: Synchronous and Asynchronous constructor

The reason why the two modes use the same constructor is because they both use the variables in order to generate the play environment. The synchronous mode however has less variables and thus can be created in a more simpler way. When generating the window, the pointers as well as the ellipses that the players have to follow will be set up. The constructor then generates a ***AnimationQueue*** class. This class is an internal class that will store a list of all the movements the circle will have to make and will ensure that the movements are done smoothly and without any rough movements. After that the timer that will send information to the ***CoordinateHolder*** class will be set up with the ***SendInfo*** method and interval of 100 milliseconds. The window also sets a binding to it's resolution so that the ellipse trajectory can be scaled to the appropriate size of the monitor it will play on. The ***MovementWindow*** will run in full screen and it was requested that the whole size of the screen to be used in this feature.

After the general settings have been made, the window will then set the course running type using the "mode" string that it will receive from the "MainWindow".

If the user has set the mode to asynchronous, the "generate" delegate will have the asynchronous method attributed. Apart form that a timer will be declared which will be responsible with the moving of a ***checkpoitEllipse*** on the map and also switching the colors of the two ellipses.

```
 private void Asynchronous()
        {
            int agregator = 0;
            int i = 1;
            double lastX = 0;
            double lastY = 0;
```

```
    while (i != coordinates.Count)
    {


        if (Math.Abs(coordinates[i].X - coordinates[i - 1].X) > 3 ||
         ↪  Math.Abs(coordinates[i].Y - coordinates[i - 1].Y) > 3)
        {
            this.queue1.queueAnimation(new
             ↪  DoubleAnimation(coordinates[i].X - 50, new
             ↪  Duration(TimeSpan.FromMilliseconds(1000.00 /
             ↪  CourseSpeed))),
                                    new
                                     ↪  DoubleAnimation(coordinates[i].Y
                                     ↪  - 50, new
                                     ↪  Duration(TimeSpan.FromMilliseconds(1000.
                                     ↪  / CourseSpeed))
                                    );
            // This if statement does not appear in the Synchronous
             ↪  method.
            if (agregator == 50)
            {
                checkPoints.Add(new Point(coordinates[i].X - 50,
                 ↪  coordinates[i].Y - 50));
                agregator = 0;
            }
            agregator++;

        }

        //Position Ellipse to the begining of the course where the
         ↪  first animation will start.

        if (i == 1)
        {
            Canvas.SetLeft(StimulyEllipse1, coordinates[i].X - 50);
            Canvas.SetTop(StimulyEllipse1, coordinates[i].Y - 50);
        }

        lastX = coordinates[i].X;
        lastY = coordinates[i].Y;
        i++;
    }
    Canvas.SetLeft(StimulyEllipse1, coordinates[0].X);
    Canvas.SetTop(StimulyEllipse1, coordinates[0].Y);
}
```

The asynchronous method uses the ***Course*** object from the model layer, to generate the AnimationQueue that will move the ***stimullyEllipse*** . The ***AnimationQueue*** will be explained in it's own section, so for now we only need to know that it makes circles move.

The "aggregator" integer is used in this method, to take very 50th position where the ***stimullyEllipse*** will be positioned to and use that value as a point where the ***checkPointEllipse*** will be moved to.

If the selected running mode will be synchronous, the window will use the "generate" delegate with the synchronous method attributed to it. The synchronous method is exactly as the Asynchronous method, the only difference being that it will not generate a list of checkpoints because the ***checkPointEllipse*** will not be used in this mode. Inserting the synchronous method here would just mean repeating the previous code snippet and would then be redundant.

When the self-paced mode would be selected, the second constructor of the window is called, together with the synchronous method attributed to the delegate.

```
public MovementWindow(MainWindow mainWindow , System.Windows.Media.Color
 ↪  color1, System.Windows.Media.Color color2, System.Windows.Media.Color
 ↪  color3 , int StrokeThickness)
        {
            InitializeComponent();
            Pointer1.Stroke = new SolidColorBrush(color1);
            Pointer1.DataContext = model;
            Pointer2.Stroke = new SolidColorBrush(color2);
            Pointer2.DataContext = model;
            lineBrush = new SolidColorBrush(color3);
            this.StrokeThickness = StrokeThickness;

            this.mainWindow = mainWindow;
            recorder = coordinateRecorder.getInstance(this);
            holder = CoordinateHolder.GetInstance();

            t = new System.Timers.Timer();
            t.Elapsed += new ElapsedEventHandler(SendInfo);
            t.Interval += 100;

            this.SetBinding(Window.WidthProperty, new Binding("RezolutionX")
             ↪  { Source = model, Mode = BindingMode.OneWayToSource });
            this.SetBinding(Window.HeightProperty, new
             ↪  Binding("RezolutionY") { Source = model, Mode =
             ↪  BindingMode.OneWayToSource });

            generate = Self_Paced;
            CourseComplexity = 0;
            StimulyEllipse1.Visibility = Visibility.Visible;
        }
```

The "MovementWindow" constructor, initializes the window components, sets up the pointers that the player will use with the colour that will be assigned to them from the menu and creates a data binding for the pointers to the model where the Wii remote coordinates are received. The pointers then move automatically every time those variables are changed. The colour of the line the players will follow as well as the thickness of it is also set here with values selected from the menu. Bindings of the window width and height property are also set here so that when retrieving the trajectory from the ***Course*** class, the values would be properly scaled to the window resolution. Finally, the "Self_Paced" method is attributed to the "generate" delegate and the courseComplexity is set to 0. The curse complexity integer is used to identify which of the 4 trajectories we want to retrieve from the ***Course*** class. The ***StimulyEllipse1*** will be show on the window but only for a brief time so that the users now from which position they should start following the trajectory.

When the experiment conductor wishes to start the experiment from his command window, the class will call the generate method, which will build the course in the selected mode and make it ready to be played.

```
public void buildCourse()
        {
            course = new Course(this);
            try {
                if (CourseComplexity == 0)
                    coordinates = course.simpleFunction();
                if (CourseComplexity == 1)
                    coordinates = course.firstFunction();
                if (CourseComplexity == 2)
                    coordinates = course.secondFunction();
                if (CourseComplexity == 3)
                    coordinates = course.thirdFunction();
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            if (generate == null)
            { MessageBox.Show(" Please select a course setting from the main
             ↪ window before attempting to run ."); }
     s        else
             {
                generate();
                built = true;
             }}}
```

The build method will make a instance of a ***Course*** object which will then use to get a set of coordinates from one of the four mathematical functions stored in the class. If any problem occurs during a message box with an error message will be printed out. Otherwise, the course is generated in whatever mode the user chose.

```
private void Self_Paced()
       {
           int i = 1;
           double lastX = 0;
           double lastY = 0;
           while (i != coordinates.Count)
           {
               Line l = new Line();
               l.Stroke = lineBrush;

               if (lastX == 0 || lastY == 0)
               {
                   lastX = coordinates[i].X;
                   lastY = coordinates[i].Y;
               }

               //Draw the line
               l.X1 = lastX;
               l.X2 = coordinates[i].X;
               l.Y1 = lastY;
               l.Y2 = coordinates[i].Y;

               l.StrokeThickness = StrokeThickness;
               StimulyReferencePoint.Children.Add(l);

               //Position Ellipse to the begining of the course
               if (i == 1)
               {
                   Canvas.SetLeft(StimulyEllipse1, coordinates[i].X - 50);
                   Canvas.SetTop(StimulyEllipse1, coordinates[i].Y - 50);
               }

               lastX = coordinates[i].X;
               lastY = coordinates[i].Y;
               i++;
           }


       }
```

When the course is built with the "Self_Paced" method, the method will go through coordinate set retrieved from the ***Course*** object, and will use those coordinates to draw a trajectory that will then be displayed on the window for the players to follow.

4.4.5 The AnimationQeueue class

The ***AnimationQueue*** class is a class that I had made specifically to animate the ***StimulyEllipse1*** on the screen for the movement task that participants are expected to perform. It is important to note that the class was not made in the main project period, but was developed during the internship period while working on this project.

The class uses two animation collections that are used to animate any object on a XY axes. One animation queue handles the Y-axes and the other handles the X-axes. In my software I needed to animate the ellipses from one point to another as smooth as possible. Doing so by simply modifying the X and Y coordinate of the ellipse was not enough as the movement became very lagged. However I noticed that by using the "DoubleAnimation" feature from the WPF framework the movement would be much smoother. The ***AnimationQUeue*** class uses the ***DoubleAnimation*** feature of WPF and makes it possible to have multiple animations set to trigger one another upon completion. This was a bit tricky to accomplish because if an animation would be triggered why the previous one was still being executed, the whole process would stop entirely. For this I made it so that once the ***DoubleAnimation*** object fires it's ".completed" event, the next animation in line will be triggered. Because this class was created during the internship period you can look further in the code in the project appendix.

4.4.6 Stimuli Window - Reading task

The reading task has been implemented almost in the same way as the movement task. The experiment conductor is able to run the reading task in three modes the same as with the Movement.

The Asynchronous reading mode will have two users read a text in a turn based manner. A portion of text will be highlighted at all times and will move across the whole text to show where the users should be with the reading so that they will have a reading paced given by the experiment conductor. The highlighted text will then alternate between three colours: one colour to indicate that the first participant is expected to read, one colour to indicate that the second participant is expected to read, and one intermediary colour to show when one participant should get ready to stop reading and when one participant should get ready to start reading.

The Synchronous reading mode will have the two users read, again the same text with a highlighted portion that will move across. The difference here is that the text will no longer switch colours. Instead participants are asked to read together the text.

The Self-Paced mode will let the two participants determine their own reading speed preference. No portion of the text will be highlighted so that they can go at their own pace.

When the experiment conductor will have made all the necessary settings in the Main Window the Reading task window will be initialized with one of three different constructors. In the three constructors, there are portions of code which are identical and then there are some which set of the reading modes.

```
public ReadingWIndow(int fontSize, string path, /*Different arguments here
 ↪  depending on constructors */)
        {
            try
            {
                this.path = path;
                this.textSize = fontSize;
                InitializeComponent();

                 StreamReader reader = new StreamReader(@path,
                  ↪  Encoding.Default, true);
                prompterText.AppendText(reader.ReadToEnd());
                prompterText.FontSize = textSize;
                prompterText.HorizontalAlignment =
                 ↪  HorizontalAlignment.Center;

                 TextPointer text = prompterText.Document.ContentStart;
                while (text.GetPointerContext(LogicalDirection.Forward) !=
                 ↪  TextPointerContext.Text)
                {
                    text =
                     ↪  text.GetNextContextPosition(LogicalDirection.Forward);
                }

               /*
               The different code for each of the costructors will be
 ↪  here,but discussed a bit later.
               */


            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Common constructor parts

Each of the three constructors will have had this segment of code but with different arguments depending on what needs to be set up. The constructors will take the path given in the MainWindow from where the text to be displayed is retrieved. A font size is set so that the text is easily readable for participants. A ***StreamReader*** is initialized so that the text can be retrieved and added in the "prompterText" container which is of time ***RichTextBox*** [9].

If the reading task will be running in self-paced mode the constructor will have one extra argument of type integer and will be called "_ScrollDownSpeed".

```
    scrollTimer.Elapsed += new ElapsedEventHandler(ScrollDownEvent);
    scrollTimer.Interval = 1000 / _ScrollDowndSpeed;
     this.KeyDown += new
      ↪  KeyEventHandler(PrompterWindow_KeyDown_OnSelfPaced);
```

The Constructor will also add a ***Timer*** object that will scroll down the text slowly in case the inputted text is large enough and a "KeyDown" event handler. The "KeyDownEvent" handler will allow allow the experiment conductor to manually scroll up or down the text in the eventuality that the participants have read faster or slower than expected.

In the synchronous reading mode, the constructor will have two extra arguments:

```
/*
Extra arguments
*/
int _TraversalSpeed , System.Windows.Media.Color color1

/*
Individual code for Synchronous mode
*/

 traversallTimer.Elapsed += new ElapsedEventHandler(TraversalEvent);
 traversallTimer.Interval = 1000 / _TraversalSpeed;
 TextPointer startPos = text.GetPositionAtOffset(index);
 TextPointer endPos = text.GetPositionAtOffset(index + 30);
               prompterText.Selection.Select(startPos, endPos);
               prompterText.SelectionBrush = brush1;
 this.KeyDown += new KeyEventHandler(PrompterWindow_KeyDown_AsyncAndSync);
```

The synchronous mode will feature a timer that will be responsible with moving the highlighted text position by an increment of one each time an elapsed event is reached. The "startPos" and "endPos" are used to specify the start and the end of the highlighted text and a key event handler is set so that the experiment conductor can adjust the speed at which the highlighted text moves across the screen.

[9]RichTextBox is a UI Element from the WPF framework that is able to contain and display text

The asynchronous mode runs in exactly the same manner as the synchronous mode except that it will feature a second ***Timer*** object that will be responsible with switching the colours of the highlighted text.

For a more detailed view on the code please see Appendix.

4.4.7 The recording feature

The recording feature of the program is split into two sections. The first section involves recording the audio input from the users in audio files. The second section is recording the coordinates of the three active elements present on the Movement task window : StimulyEllipse, Player1 pointer, and Player2 pointer.

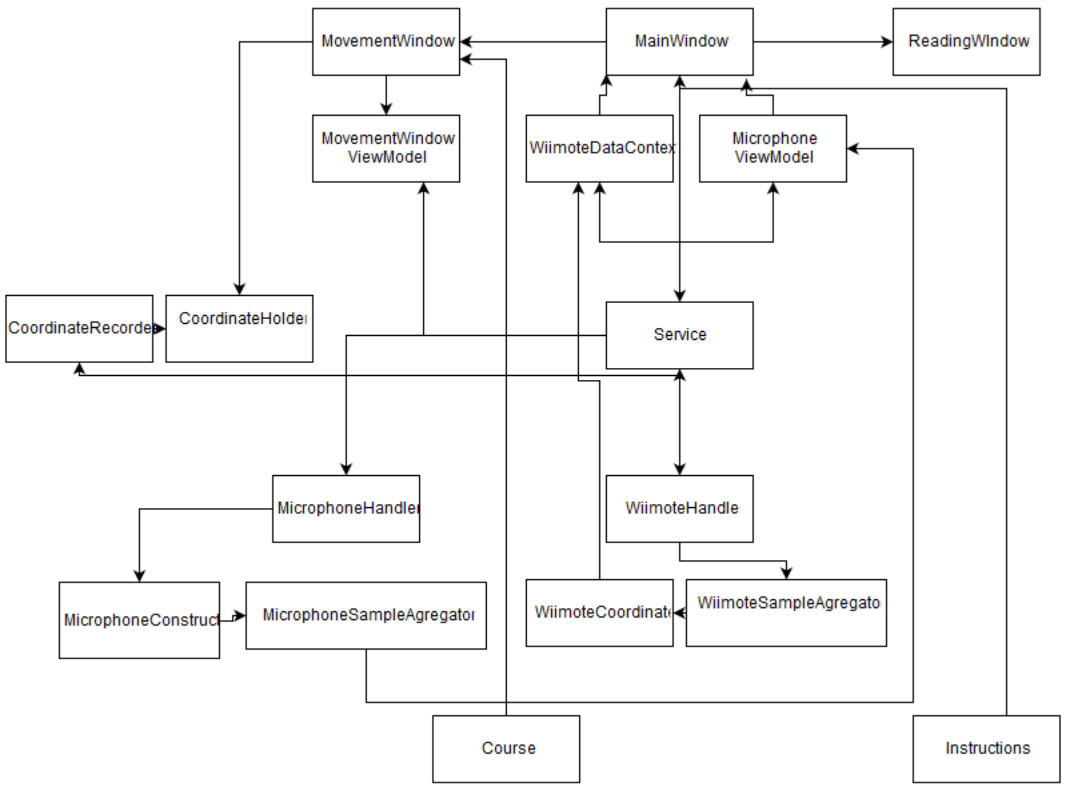
The Audio recording feature is part of the microphone handling classes. Each time a microphone is expected to record audio, a WAV file for this microphone is created in the audio recordings folder created by the ***MicrophoneHandler*** class. Software will then record any audio sample that comes from the microphone into that specifically audio file until the recording is ordered to stop.

The coordinate recording feature has it's own classes defined. This came as a necessity in order to avoid any threading conflicts that might have occurred between the GUI thread and the timer thread that was tasked with saving the data every 1/10th of a second in the session defined file.

4.4.8 Sequencing feature

The sequencing feature, was requested in the last weeks of the project. Because of this not much progress has been made on it yet. The Sequencing feature, is intended to allow the experiment conductor to plan the order in which both tasks will be done by the participants, as well as select the priority of the 3 running modes. Currently the feature is able to create a sequence that will be executed, however further investigation and testing is required to ensure that the feature performs as expected in all circumstances as well as being able to record the expected data in the same manner.

4.4.9 Final project architecture



The architecture of the project has mostly kept it's initial composition. The classes all being organized in a 3 layer architecture.

The Model layer is composed of the microphone components and Wii remote components. Although the composition of the microphone and the Wii remote classes have changed significantly from the initial design. The microphone and Wii remotes send information to the GUI layer's through the ViewModel classes that were created to take their specific input.

The Service layer acts as a intermediary between the GUI and the Model layer. In the initial stages of the runtime, the Service class will have the responsability of linking the Model layer with the GUI layer. After the initial run-time the service is used to manage these connections and send out commands to each individual component. The Coordinate recorded is listed as being part of the serice class. This is because the coordinate recorder component is used only by the MovementWindow directly.

The GUI layer made up of the ***MainWindow, MovementWindow*** , and ***Reading-Window*** together with the ViewModel classes handles all the graphical and user experience tasks in the software. These include the setting up of the session by the experiment conductors in the MainWindow, and allowing experiment participants to interact with the software through the use of the MovementWindow and ReadingWIndow. Part of the interaction is also handled by the Microphone and Wii remote handling which handle the input from the experiment participants.

The architecture was built and focused around the use EventHandlers as the main source of comunication between layers and classes. Because of that reason, the class relationships do not adhere to a single pattern such as MODEL-SERVICE-GUI information flow which is standard for a data management application. Instead the architecture allows for class relations to look past layers so that code performance and volume stay as low as possible.

4.5 Description of the process

Throughout the development of my project, the process that I had used in my sprints for developing the features had gone through some changed. Some changes were done because they were completely necessary, while others I had found to be a better addition and improved the overall sprint performance.

Each sprint would start out with a Monday Scrum meeting. Every second Monday, the whole office: me, Dan Monster, Joshua Brain, Rene Frederiksen and Michael Futtrup would spend half an hour to an hour and discuss what we will be doing in the sprint that would start then and end in two weeks on a Friday. Throughout the week each of us would focus on our individual projects and at the end of the week on Friday, we would have a informal discussion about any developments or problems we had stumbled in during the week. This meeting usually took place on Friday mornings between the three interns: me, Rene and Michael. The second Monday of the sprint we would have a update meeting consisting of the whole office again and talk about what we had to do in the week and how we planned to achieve our goals. The last day of the sprint was always a Friday. Friday was a no-commit day, a day which I reserved for the assessment of my performance throughout the whole sprint duration and make any decisions to apply to my next sprint such as adding more time for each backlog task, or more frequent checking of my WakaTime work report.

My first sprint, which aimed at improving the Wii remote feature had a straight forward approach. The whole sprint was done in a whole linear motion of Research-Implementation-Testing. For each task on the sprint backlog, I would first research into the matter whether it was a bug that needed fixing or a new functionality that needed to be added. After I would find viable alternatives I would then start implementing the one that was the most favoured. If the feature performed as expected, then I would move on to the next task.

In the sprint planning, I did not do any meticulous estimate. Because I had previous experience in working with the WiimoteLib, I did a rough estimate and felt confident that in the two weeks that the sprint will last, I had enough time to solve all the tasks. While this proved to be true, and I did have enough time to do all the tasks, I felt that my productivity and the quality of the software had suffered because of my lack of estimates. Not only that but I was not entirely sure that my shortcoming in software quality did not endanger the outcome of the project in any way. The tasks that I did not manage to finish in their planned sprint ended up being pushed a different sprint at the end of the development.

Gradually, with each passing sprint, I started estimating my tasks more and more accurately. Eventually I noticed that, by estimating the task required time as precise as possible, my productivity grew and so did the software quality. In my third sprint, which focused on the microphone features, I was able to finish all backlog points and had time to work on some existing bugs.

Backlog	Estimated time	Finished in time	Finished
Architecture design			✓
Implement WiimoteHandler			✓
Implement WiimoteCoordinate			✓
Implement WiimoteSampleAgregator			✓
MVVM pattern for remote interface			✓
Calibrate for different screen sizes			
User Testing and debugging			

Figure 6: First Sprint backlog

Backlog	Estimated time	Finished in time	Finished
Volume feedback	2 days	✓	✓
Available microphones	2 days	✓	✓
Recordings possible	1 day	✓	✓
MVVM pattern	1 day	✓	✓
Debug audio	not estimated		✓
Invisible char bug in prompter(pushed from second sprint)	1 day		
Intermediary color between switches(pushed from second sprint)	1 day		

Figure 7: Third Sprint backlog

By comparing the backlogs of my first and third sprint and looking the points of the initial backlogs I could see a significant improvement in my work. In the estimation part, a important role was also played by the WakaTime API mentioned in the beginning of the report. By analyzing the feedback from the API I could determine that my peak performance was always on Mondays and Thursdays. Using that information I programmed my work schedule around those two days.

4.6 Description of the product - Visual User guide

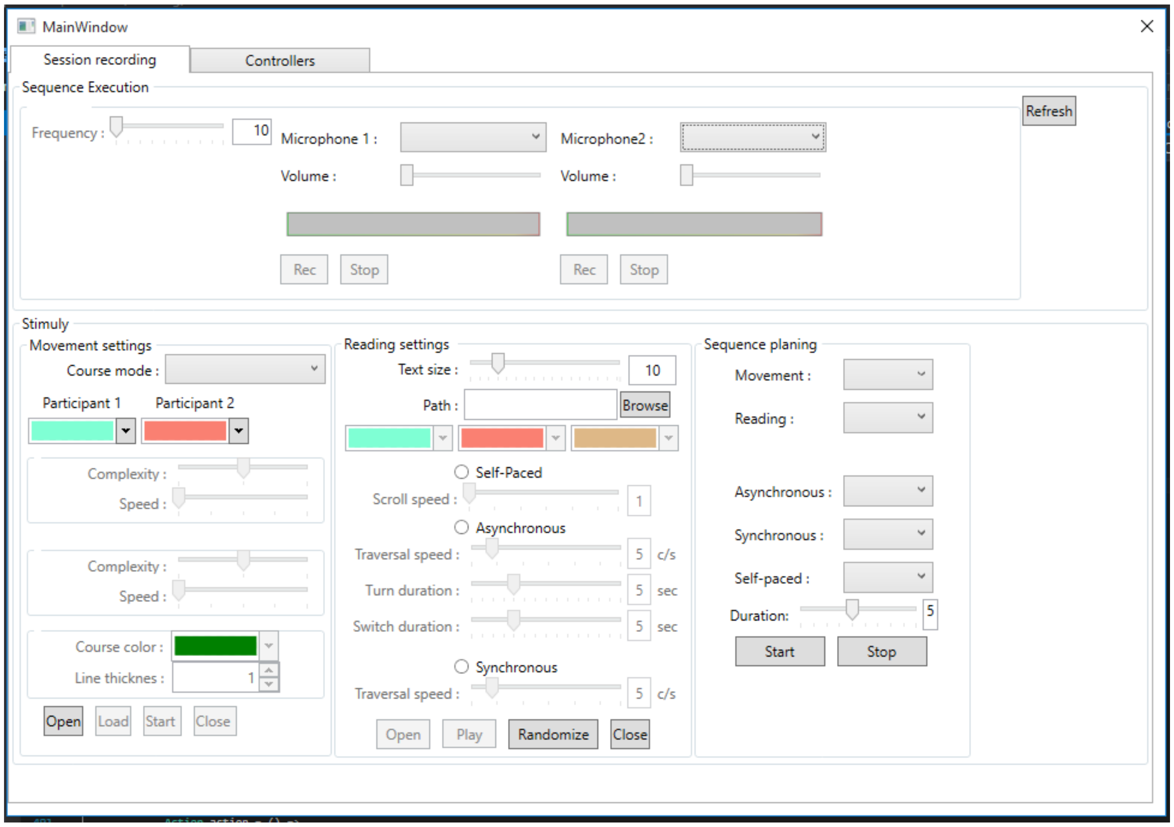


Figure 8: Main Menu

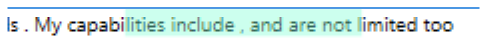
The main menu of C.O.B.E.S. is mane out of 4 sections, each section controlling a different feature, and a tab menu to alternate between the main menu and the Wii remote menu.

The "Sequence" section of the Main Menu, is used to make use of the Sequencing section. Unfortunately as this feature was a relatively new requirement that was made in the past month, not much has been achieved in it's development and is still being worked on.

The "Movement Settings" section, allows the experiment conductor to select with the combo box, which Course mode he wants to run in the experiment. When selecting on of the option in the combo box, the settings for that respective mode will be enabled in the menu.

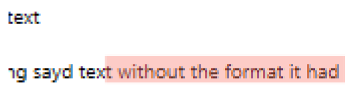
The experiment conductor has the option to chose the speed at which the "StimullyEllipse1" will move on the screen and also the complexity of the trajectory that it will follow. If the selected mode is Self-Paced however, the course color option will be made available as well as the thickness of the line that will be displayed on the screen. Here the color of the two pointers that will be displayed on the screen can also be changed and that will also change the color of the two spheres.

The same principles are used in the Reading Settings section. Here the font size that will be displayed on the prompter ca be changed to suit various screen sizes. As a general observation, in the manner that the experiment is planned to run the text is always set at the maximum size. With the use of a dialog box from the WPF framework, the text that we want our participants to read can be selected, as long as it is in a .txt format. The colors that will be used for highlighting the text are also up for choosing.



ls . My capabilities include , and are not limited too

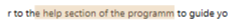
Figure 9: First color



text

ng sayd text without the format it had

Figure 10: Second color



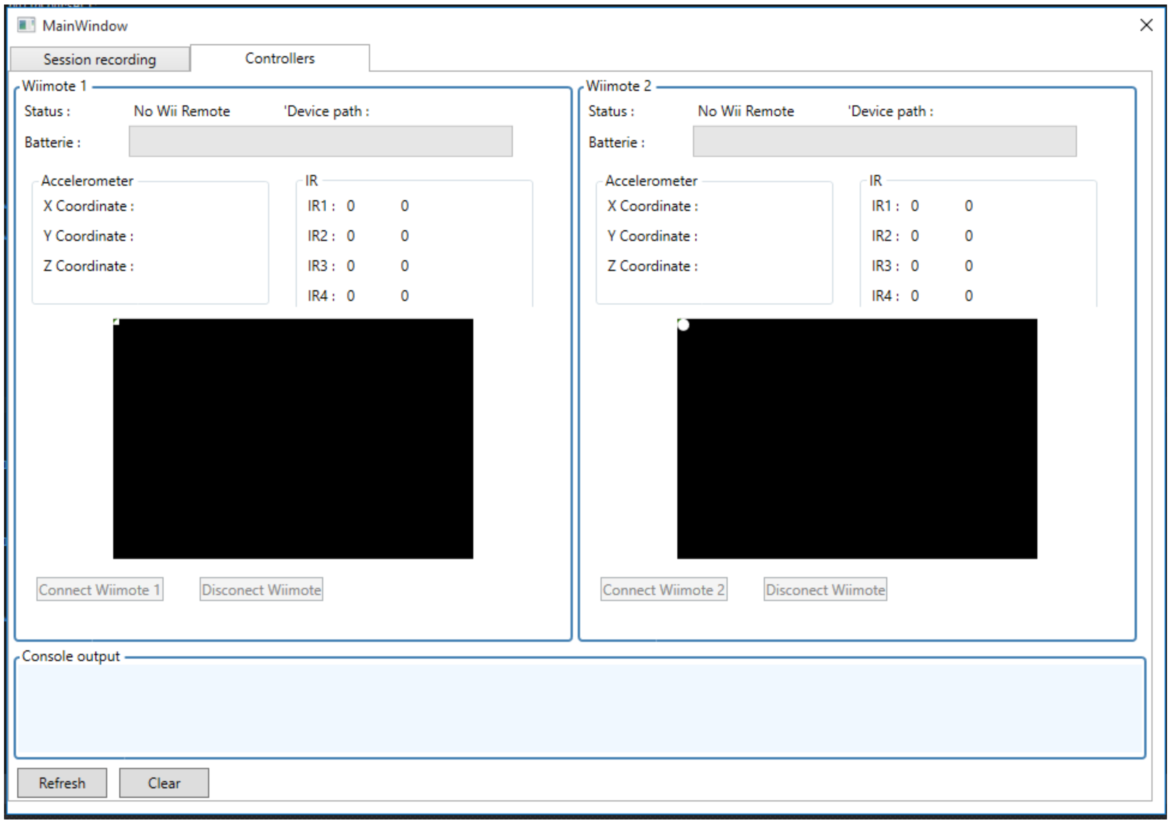
r to the help section of the programm to guide yo

Figure 11: Third color

The first color indicates when the first participant is expected to read. The second color indicates that in a short time the second participant will have to read. When the highlight of the text switches to the third color, then the second participant starts to read. The process repeats itself backward and forward.

The speed of the highlighter that is running along the text can be adjusted from the menu as well as the time intervals between the participants turn to read. If the experiment is running in Synchronous mode than the text highlighter has only one color through out the whole duration and in Self-Paced mode there is no text highlighter.

In the microphone section, the conductor can which microphones will be used for audio input. If the microphones has already been selected the software won't allow it to be selected for both the selection slots. If the microphone will be active the volume will be displayed in the horizontal bar as feedback. The "Refresh" button retrieves a new list of available microphones from the system, in the case any modifications have been made to the hardware setup.



Wiimote 1

Status : No Wii Remote 'Device path :

Batterie :

Accelerometer

X Coordinate :

Y Coordinate :

Z Coordinate :

IR

IR1 : 0 0

IR2 : 0 0

IR3 : 0 0

IR4 : 0 0

Wiimote 2

Status : No Wii Remote 'Device path :

Batterie :

Accelerometer

X Coordinate :

Y Coordinate :

Z Coordinate :

IR

IR1 : 0 0

IR2 : 0 0

IR3 : 0 0

IR4 : 0 0

Figure 12: Controller menu

The Controller menu is where the Wii remotes can be connected to C.O.B.E.S. The user clicks the refresh button on the button, which will retrieve a list of all Bluetooth devices that are connected to the computer. In the console output, a message will appear informing the user about how many remotes are connected to the device. When the user wants to connect a remote to C.O.B.E.S. he will use the "Connect Wiimote" button which will establish a connection to the desired remote.

If the connection was successful, the coordinate values of X, Y and Z accelerometers will be updated as well as the values for the Infrared(IR) LED's. In the dark panels the IR coordinate values will be displayed in the form of dots, one for each LED. In this way, a digital representation will be made of what the remote can "see" with it's IR sensors.

Once all the settings have been done and the remotes have been connected, the movement task of the experiment can be performed in all the three modes.

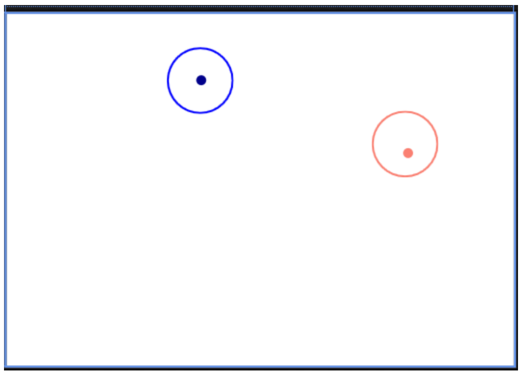


Figure 13: Asynchronous mode

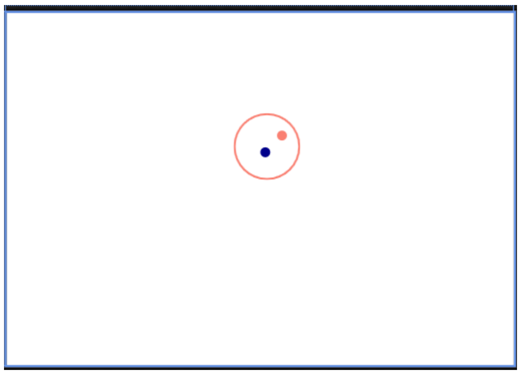


Figure 14: Synchronous mode

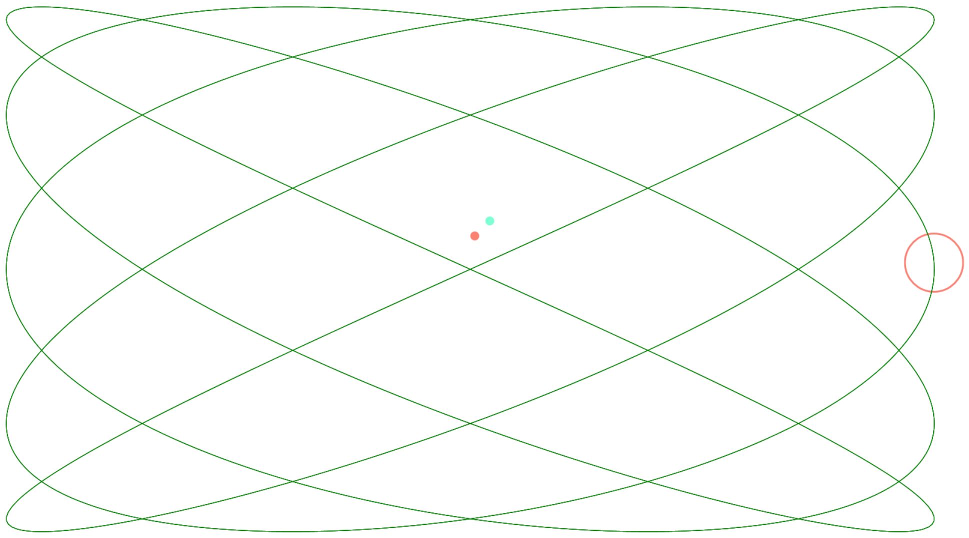


Figure 15: Self-paced mode

5 Reflections

After finishing the project and using the report writing as a opportunity to look back and reflect on the various aspects that influenced the final outcome of C.O.B.E.S., I was able to have a more impartial view of what processes could have been performed me efficiently and which might not have been needed at all.

5.1 Reflections on chosen methodology

When it comes to the choosing of methodology, I believe that Scrum was good choice. By following the principles and work patterns of Agile and Scrum methodology, I was able to quickly learn and apply new principles and tools in the project. The decision to adapt to the current working context and environment proved to be a good alternative compared to attempting to impose a certain work method. Scrum methodology ended up being a well suited for my given predicaments such as the team size, available time, and availability of project owners. The work environment was easily maintained and at no point was there chaos or uncertainty in the development plan.

The developer-project owner relationship had high benefits from the use of the agile principles. By involving the project owner in various processes and keeping him up to date on a weekly basis, a healthy work relationship was formed that benefited the project development as well as the overall morale of the team.

Looking back, I believe that the level of personal involvement of each party in the project made for an appropriate development environment. Both me(developer) and Dan Monster(project owner) were fully committed to the project. For me(developer) the project represented a opportunity to assert my ability as a software developer. For Dan, the project possibly meant the last opportunity to conduct the planned research as Aarhus University funds will be made unavailable for this study in the year 2016.

If an alternative methodology should be taken into consideration for the project, I believe the Extreme Programming methodology to be a good option as long as several conditions are met. The first condition would be to eliminate the need for researching tools. Extreme programming is more plan driven and because of that there is no way of knowing if you will meet the time requirements since research can take an indefinite amount of time. A bigger team is also needed in order for Extreme programming to be a viable alternative. Being a small team of 5 developers could possibly mean that not all the managements tools advocated by Extreme programming are needed. In this case we come back down to the Scrum methodology.

5.2 Reflection on product and process

The process overall suited the project and what it hoped to accomplish. It left room for unforeseen delays as well as research time. With that said, I do believe that the process could have benefited from a more rigorous planing and sprint analysis. The planing Mondays of every sprint should themselves have a backlog of what is it that has to be planned and accounted for. On the sprint analysis, the main criteria that I used for success measurement was the weather or not the feature that was developed could be considered release worthy.

While this did not have a negative impact on the process and development, it did not offer much feedback in terms of what could be improved in the following sprint and I had to look into other aspects such as effective work time, research time and overall final quality. One managing tool that I felt missing from the process was the use of visual tools such as managements charts and scrum board. In the process I have kept at all times notes about my sprint planing and backlogs as well as all the steps I have taken such as the decisions between different external libraries. These notes however were most in plain text. For myself as a more of a visual learner I think having these in a physical scrum board could have been a good idea for keeping an overall constant productivity.

5.3 Reflections on the teamwork and distribution of roles

Throughout the project, the development team had always been comprised by myself and no one else. This did not prove to be a problem as the size and requirements of the project were adequate for time frame that I was given. Being a one man team however gave me some insight into the importance of separating various tasks into teams and also the importance of making it possible for the teams to work together as easily as possible

The best example of this case that comes to mind is the initial and final architecture of the GUI - Model interaction. While it is true that the changes were done in order to improve performance and meet UX requirements, it also showed the importance of tasks delegation. Before my implementation of MVVM patter, if the GUI layer were to be developed by a whole team of UX developers they should have had knowledge of the internal workings of the Model layer. Either that, or the back-end developers should now how the GUI layer works. That would have proven to be big amount of time spent on debriefing each teams on this subject. By the use of the MVVM pattern however, a proper distribution of roles could be done between the two teams with as little concern for the other teams progress as possible.

Being a one man team meant that I had to fulfill several roles such as UX developer, Software developer, Architect, and project manager. This was not an easy task as at times I found it difficult to focus on one task because I was still thinking on a previous task that I had to do or one that will come shortly after. Having a distribution of roles properly done would allow each team member to focus on his tasks alone and trust that his other colleagues are doing the same.

6 Future development possibilities

Through out the development of o the project, there have been several ideas that came to mind, ideas of features for the project that could possibly extend it's uses to more than what it has been planed.One such idea is to make the whole software modular.

The idea of a modular C.O.B.E.S. refers to possibility that instead of having hard coded stimuli that can not be changed in the program, the researchers would have the possibility to easily create a script that would generate the required stimuli automatically. A different possibility to achieve this was to have C.O.B.E.S. run a executable file that would present the experiment participants with various stimuli. In this scenario C.O.B.E.S. responsibility would be to manage the executable files as well as the interaction with various devices. This feature however would require that the Wii remotes to be treated as regular controllers by the machine so that their input could be used in the executable that is running from C.O.B.E.S. and so the Infrared features would be lost.

7 Conclusion

The project has developed a software solution for Cognition and Behavior Lab that addresses all the initial requirements that were established in the project genesis as well as several requirements that have been made during the development phase. Several meetings have taken place throughout the whole project to ensure that the direction of the development and the vision for the project is the same one that the researchers had for their experiments.

The software however is not yet ready for release as there are still issues that need to be addressed, Issues that concern the overall user experience. Seeing as the experiment does not have a fixed start date I plan to request permission from Dan to keep working on the software until it is ready for release.

References

(2015), *WPFToolkit*. `http://wpftoolkit.codeplex.com/`.

Brian Peek (2009), *WiimoteLib v1.7*. `https://wiimotelib.codeplex.com/`.

Dan Mønster (2014), *CobeLab Resources*.

Dan Mønster, Kristian Tylen , Riccardo Fusaroli (2014), *Heart Rate Synchronization in Social Interactions*. `http://pure.au.dk/portal/en/projects/heart-rate-synchronization-in-social-interactions(53edc51d-785b-439f-be2e-db46a6b62b6b).html`.

Johnny Lee (2008), *Wiimote Ted Talk*. `https://www.ted.com/talks/johnny_lee_demos_wii_remote_hacks`.

Mark Heath (2015), *NAudio*. `https://naudio.codeplex.com/`.

Schwaber, K. (1995), *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*.

Schwalbe, K. (2012), *Proceedings of the Information Systems Educators Conference ISSN. 1435*. Managing a project using an agile approach and the PMBOK® Guide.

Smith, J. (2008), *WPF Apps with the Model-View-ViewModel Design Pattern*.

Yousra Abdo Hardb , Cherie Noteboom , Surendra Sarnikar (2015), *Evaluating project characteristics for selecting the best-fit Agile Software Development methodologyL: A teaching case*. Journal of the Midwest Association for Information Systems (JMWAIS): Vol. 1: Iss. 1, Article 4.