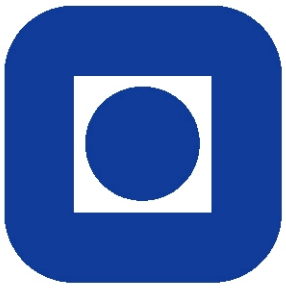
SPEECH AND MOTION DRIVEN UX

PROJECT REPORT
TDT4290 CUSTOMER DRIVEN PROJECT
CUSTOMER: ITERA CONSULTING

GROUP 6

Mirna Besirovic
Joachim Halvorsen
Eirik Mildestveit Hammerstad
Mahboobeh Harandi
Johanne Birgitte Linde



Norwegian University of Science and Technology, November 2012

Abstract

Speech and motion driven UX is a project consisting of making a 3D model, controlled by motion and voice, using a Kinect. The scenario is to show the airspace with airplanes as a 3D model. The reason for making such a model is to try to integrated the motion interpreting functionality of the Kinect in a 3D dynamic environment, and also to use its voice recognition ability in the same environment. The motion recognition is used for controlling the viewpoint of the user, while the voice is used for controlling the airplanes. Such a system might be useful for solving the problem with too few air traffic controllers, avoiding loss of incomes due to cancelled or delayed airplanes. The resulting system was built focusing on requirements regarding usability, modifiability, testability and performance. MVC was chosen as the architectural pattern. By using Scrum and test driven development, we managed to create a prototype of the system with motion and speech controls.

This task was given at the master-level course TDT 4290 Customer Driven Project, at the department of computer and information science (IDI) at the Norwegian University of Science and Technology (NTNU). The project was carried out during the autumn of 2012 for the customer Itera Consulting.

We would like to thank our customer contacts Mário Vaz Henriques and Karl-August Brunstad at Itera Consulting for their help and feedback during the project. We would also like to thank our supervisor Meng Zhu for his help and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Speech and Motion Driven UX (User Experience) is one of the student projects executed during the master-level course TDT4290 Customer Driven Project, which is a software engineering course provided by IDI, NTNU. The goal of the project is to teach the students the art of working in groups where they have to use our software engineering skills in the context of a development project, given by a real-world customer. In our case the customer was Itera Consulting.

In this chapter we present some general information about the project, including the background, the stakeholders and the demands from our customer. To provide the reader with an overview of the project, this chapter also covers the main aspects of the report and its contents.

1.1 Project Description and Background

The customer desired a system, where a 3D model could be manipulated using speech and motion. The scenario of focus should be an air traffic controller environment, where the system is imagined to be used when guiding airplanes during landing and takeoff. The system should also provide the air traffic controller with an overview of where the planes are located relative to each other. The user should be able to rotate and move around in the air traffic area with motion, and be able to redirect the airplane's position and speed using voice control.

The system is only intended to be used as a tool for testing the technology and will only be a prototype. It is also intended to be used only by one user at a time, at only one computer, thus there will not be any server in our system.

In the summer of 2012 Gardemoen airport had a problem regarding the air traffic controllers [25]. The airport can accommodate up to 60 aircraft per hour, but due to short handed staff that summer, they were down to 20 aircraft per hour. Our system might be a solution to consider in the future, as it might provide a more efficient environment for an

air traffic controller, requiring fewer staff members to do the same job as before.

For the customer, the intention of the project was to get in touch with bright students at NTNU and to get an insight in to the level of knowledge. They also wanted a system that combines future technology together, so future clients could see the possibilities.

Itera Consulting has earlier worked with a similar project in a statical environment. In this project they wanted us to test the opportunity for use of the same technology in a dynamical environment. The project they had worked with earlier was a system that used the Kinect to make "art" on t-shirts. The user threw imaginary paint at a screen, showing a t-shirt, to make a pattern. The colour desired for the pattern is chosen by a speech command. This pattern can later be printed on a real t-shirt. This was a very simple system, so the customer had a desire to expand the use of the Kinect to a more dynamical environment with the use of 3D and voice commands.

1.2 Goal of the Course

The goal of the course is to teach software engineering skills in the context of a development project. The project deals with real customers, making the situation more like the ones we will experience when working in the IT consulting business.

The focus on the project is the early phases in development, i.e. project planning, pre-study, requirement specifications and system design.

At the end of the project the team will deliver a project report and hold a presentation and demonstration of the system. At this time, the system should be a finished running product. The course grade will be given based on the documentation, the final product and the presentation.

1.3 General Terms

The customer had no special demands that lead to limitations in the group. However they expressed a desire that Scrum should be used as the methodology and C# or other Microsoft-based language should be used as the programming language. This was because Itera Consulting (our customer) mainly works with Microsoft-technology and is using Scrum. The group was also given a Kinect, to allow us to test our system and become familiar with the Kinect and its use.

Relating to the time set aside for this project (3 months) there was a lot of work that needed to be done. Based on the complexity of the task and the experience of the team it was required that we use a lot of time on the project.

Duration

The project was introduced on August 21st and delivered on November 22nd.

Schedule of Results

We used Scrum as our development method, a discussion about our choice can be found in section 3.1. At the end of each sprint there were some deliveries. Each delivery consisted of a working version of a part of the system, demonstrated to the customer.

Major deliveries

In this project we had some major deliveries. They were the preliminary report, the final product and the report, and a presentation of the finished project. For a better overview of the deadlines, see section 2.2.1

1.4 Involved Parties

The project involved three different parties. The customer was the one who provided us with the task, the supervisor who helped us with group dynamics and administrative questions, and the development team, the group. In addition to the end users, these involved parties were also the stakeholders for the project.

1.4.1 The Customer

The customer was Itera Consulting, with Karl-August Brunstad and Mário Vaz Henriques as the customer's representants for this project. Their contact information is shown in section 1.1.

Itera Consulting is an IT-consultant company that delivers services and business solutions to Scandinavian businesses and organizations [55].They have deep industry knowledge mainly in banking, finance and insurance, but are also active in sectors like IT and tele commerce, health, oil, service and transport, to mention some.

The company has 15 years of experience in the IT scope. They have offices in four different countries: Norway, Sweden, Denmark and Ukraine. In Norway they started out as Objectware in 1995, before they in 2000 became a part of the Itera Group. In 2010 they extended their activities and skills, and renamed to Itera Consulting.

Itera Consulting focuses on four main values: entrepreneuring, teamwork, results and passion. Their slogan is "*Making a difference. Always*". They strive for creating good products that make a difference. Now, Itera Consulting are specialised in many fields, including launch portals, business intelligence, flexible development in Microsoft .NET and Java, project management and application management. This broad competence makes them able to be a provider of all the IT solutions of a customer and to follow up their systems through their whole life cycle. They deliver mostly systems to Norwegian customers.

Table 1.1: Contact information

Customer Contact Information

Name	**E-mail**
Karl-August Brunstad	Karl-August.Brunstad@iteraconsulting.no
Mário Vaz Henriques	Mario.Vaz.Henriques@iteraconsulting.no

Supervisor Contact Information

Name	**E-mail**
Meng Zhu	zhumeng@idi.ntnu.no

Student Group

Name	**E-mail**
Mirna Besirovic	besirovi@stud.ntnu.no
Joachim Halvorsen	joachha@stud.ntnu.no
Eirik M. Hammerstad	eirikmh@stud.ntnu.no
Mahboobeh Harandi	mahboobh@stud.ntnu.no
Johanne Birgitte Linde	johanneb@stud.ntnu.no

1.4.2 Student Group

The development team consisted of 5 students. Four of the students were currently studying computer science (siv.ing.), while the last one was a foreign student taking a masters degree in information systems, with background in computer engineering.

The four members studying for the siv.ing degree had experience in Java programming from NTNU, while the last one had C# experience.

Beyond this, some of the development team members had earlier experience in other programming languages like C++, Python and Ruby, to mention some, from earlier courses at NTNU or summer jobs. Two of the members did also have practical experience in agile development, as they had used Scrum in summer jobs.

See section 1.1 for contact information.

1.4.3 Supervisor

Our supervisor was Meng Zhu, and he was a "guider" in the project. His responsibility was to keep an eye on the main process of the group's work, and to make sure we had sufficient contact with the customer. The supervisor had also the task of following up that the group wrote the documentation in a satisfying manner, and gave guidance if necessary.

To be able to do all this, the supervisor needed to receive updates regularly about the status of the project, copies of relevant work plans and technical documents. All of this were then discussed in the supervisor meetings, held once a week.

See section 1.1 for contact information.

Table 1.2: Stakeholders

Stakeholders	**Description**
Development team	The creators of the system. Their interest includes planning, implementation, design and testing. The development team is interested in making the best system possible.
Customer	The customer is the one who has desired the system. The customer wants a finished product that fullfills his requirements.
Supervisor	The supervisor is a "guider" in the project. The supervisor helps the group if internal or external conflicts arise or if the group has some problems with its progress.
End user (tester)	In our project the end user is a testing person because the program is going to be used for testing. (and playing)

The stakeholders of the system were all the persons who were involved in or had interests in the system in some way. The system's stakeholders are shown in Table 1.2.

1.5 Report Outline

The different chapters and their main contents are described in this section.

Chapter 1: Introduction

The first chapter contains an introduction to the project and an overview of the report in general. In this chapter we do also introduce the course, the involved parties and the stakeholders.

Chapter 2: Project Plan and Management

The project plan and management chapter contains information about how we handled the administrative part of the project, including assigning roles and deciding on the internal structure of our group. This chapter also contains a section about risks, and a section about project scheduling.

Chapter 3: Preliminary Studies

This chapter gives an overview of what we read about and learned during our preliminary studies period. It also contains some discussions about why we ended up with the technologies we chose.

Chapter 4: Requirement Specification

This requirement specification chapter contains all of the project's requirements, including both functional and non functional requirements. There are also a section regarding use cases, user stories and personas. At the end of the chapter, the product backlog is presented.

Chapter 5: Overall System Design

This chapter contains a description of our overall system design, different patterns used, and the different views using the 4+1 model. It has also a discussion about the architecture of the system, and why we chose the one we did.

Chapter 6: Overall Test Plan

This chapter gives an overview of the tests we planned to conduct.

Chapter 7 to Chapter 11: Sprints

The sprint chapters contains information about how we conducted our five sprints, with planning, sprint breakdown, testing, results and the sprint retrospective. These chapters are 7, 8, 9, 10 and 11.

Chapter 12: Testing

Our testing chapter has a section with information about testing in general, followed by sections on how we conducted the tests in our project. It also says something about how we made sure that our system fulfilled all of the requirements specified in chapter 4.

Chapter 13: Further Work

The further work chapter is an overview of different things that could be done with the system to extend it further.

Chapter 14: Evaluation

In this chapter we made an evaluation of how we worked together in the group, what we have learned during the project and what changes we would have made if we should do it all from scratch again.

Chapter 2

Project Plan and Management

In order to have a well functioning group and a good group dynamic, it is important to define and assign different roles or responsibility areas to different people in a project. It is also important to have a clear idea of what kind of risks one can encounter in a project and which measures that need to be carried out in case of an incident.

This chapter contains information about the administrative part of the project, mainly about how the project was carried out, how the internal structure was and which roles were represented. Further, it also explains how we ensured that a certain quality was sustained, which risks we could face and the project's schedule.

2.1 Project Organization

This section consists of a description of the different roles in the group and who was assigned which role.

Our team consisted of 5 students. We decided to have a flat structure in our group, where everyone's opinion was equally important. The reasons for this were that we felt that it was important that everyone could come forward with their opinions on important questions, and everyone should contribute in the decision-making processes. Also, since we were a relatively small group, consisting of students with more or less the same experience, this type of structure was well suited. By having a flat structure all of the group members were encouraged to contribute in the decision-making processes. This type of structure also made it easier for the different members of the group to learn to cooperate efficiently as a team.

All the roles were *areas of responsibility*, where the responsible for each role had the "authority" to delegate tasks to other group members. The roles were divided between the group members at one of the first meetings. This was done by asking each person what role they wanted, and thereafter decide who should have each role. None wanted the same roles, making the distribution process easy. The role division is presented in Figure 2.1.

There are only a few exception for when some group members may have more authority than others. For example in situations where the group can not come to an agreements and a decision needs to be made quickly, the project manager has the authority to make the final decision.

It is important to specify that Figure 2.1 is not displaying the structure for the group and other participants, but rather how the involved parties are interacting. The arrows represent the communication flow between the group, the customer and the supervisor.

We also had backups for each role. In case the one responsible for a task was not present, the person having the role as their "extra" would take over. In Figure 2.1 the backup roles for a person are represented with light grey colour and are written in italic.

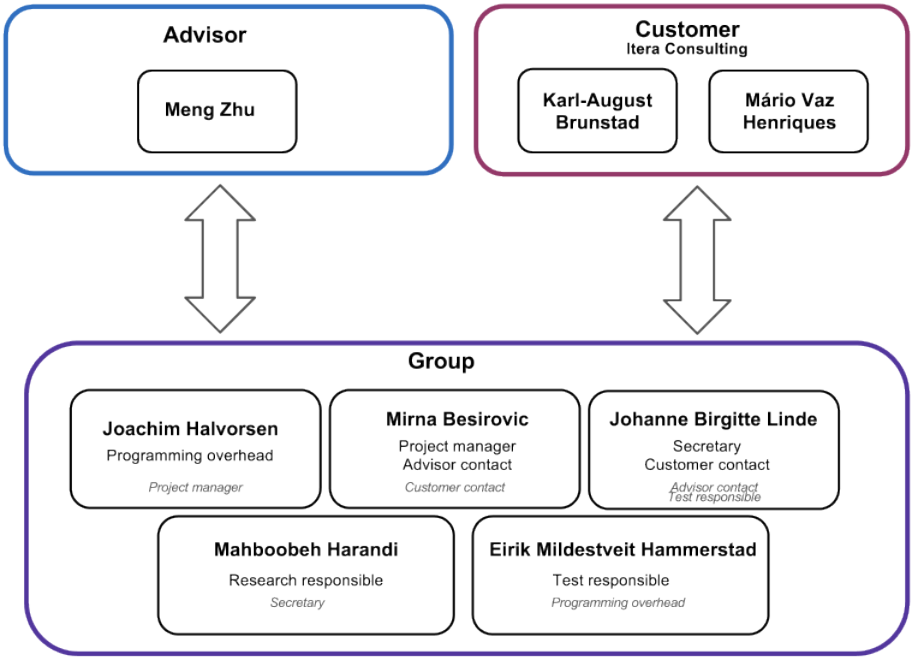


Figure 2.1: Organization chart

Project Manager - Mirna

The project manager's main task is the administrative management, including booking rooms, dividing tasks and having an overall overview of the system. The project manager must know what everyone is doing, and make sure that the schedule is followed. If there are discussions where the group does not reach a final agreement, or discussions have been too time consuming, the project manager is the one who has to cut through and make the final decision.

Customer Contact - Johanne

The customer contact is the one who communicates the most with the customer. If the group has questions for the customer, the customer contact is the one who is responsible to contact the customer and get the answers. Also, if the group has made a decision that the customer should know about, the customer contact must inform him about that.

The reason for having a customer contact is that it is much easier for the customer to only have one person to deal with, instead of a whole group. It is also an insurance that the customer will get answers to his questions.

Programming Overhead - Joachim

The programming overhead must be updated on the development of the system at all time. He is responsible for keeping track of what everyone is doing in the code, the problems they encounter and the status of all parts of the system (on time, finished, late, etc). The programming overhead will also have the final call when deciding on more code technical questions, like what language to use, details in the architecture, conventions to follow, etc. He must also make sure that everybody follows the chosen conventions.

Test Responsible - Eirik

The test responsible is responsible that the tests satisfies all the system requirements. He does also has a responsibility when new code parts are added to the system, and must make sure that these are thoroughly tested, so that no new errors are introduced. The test responsible must be able to tell the other group members if something has to be redone, instead of just letting bad code pass.

Research Manager - Mahboobeh

The research manager is responsible for finding information about the technologies and methods we are going to use. She can divide the work between the other group members, but she has the final responsibility that everybody is updated about the technologies and methods.

Secretary - Johanne

The secretary is responsible for writing summaries after each meeting with the supervisor and the customer. This minutes shall be sent to all the meeting participants, and also as a summary to those who should have been on the meeting, but for some reason was prevented from showing up. She also has to write the minutes if something important has been decided during a work session with the group.

Supervisor Contact - Mirna

The supervisor contact is the one who contacts the supervisor if someone in the group has any questions for him. She is also the responsible one for sending information about the weekly meeting between the supervisor and the group to all parties.

Scrum Master - Rotating

The scrum master is the one leading the daily scrum meetings during the sprint(s). He ensures that the Scrum process is followed, and is the moderator of the sprint reviews and sprint planning meetings. We decided to rotate the scrum master role, the scrum master is specified in each sprint. More information about Scrum can be found in section 2.3.

2.2 Project Work Plan

This section contains information about how we used Scrum in our project and an overall time schedule, including time division between tasks. We have also explained the five milestones we have set up for the project.

2.2.1 Project Schedule and Milestones

The 13 weeks were divided like shown in Figure 2.2. The Gantt chart starts at August 21st, when we had our first customer meeting and were presented with the task. The chart ends with the presentation, at November 22nd.

The chart gives an overview of the most important tasks. At the beginning of each sprint, we would break the work further down according to our backlog, but the chart would have become far too detailed if all the small work packages should have been displayed here.

Milestones

We have five milestones scheduled in our Gantt chart, Figure 2.2.

Preliminary Delivery: The first milestone is the preliminary delivery. This is a delivery of all of our finished documentation, containing an overview of all of the chapters that will be included in the final report. This milestone is scheduled to the 15th of October. The reason for this date is that it was set by the course administration, and it is important that we do not postpone all the work to the final weeks of the project.

This milestone shall be reached by starting the documentation early, and work thoroughly with it throughout the whole project period.

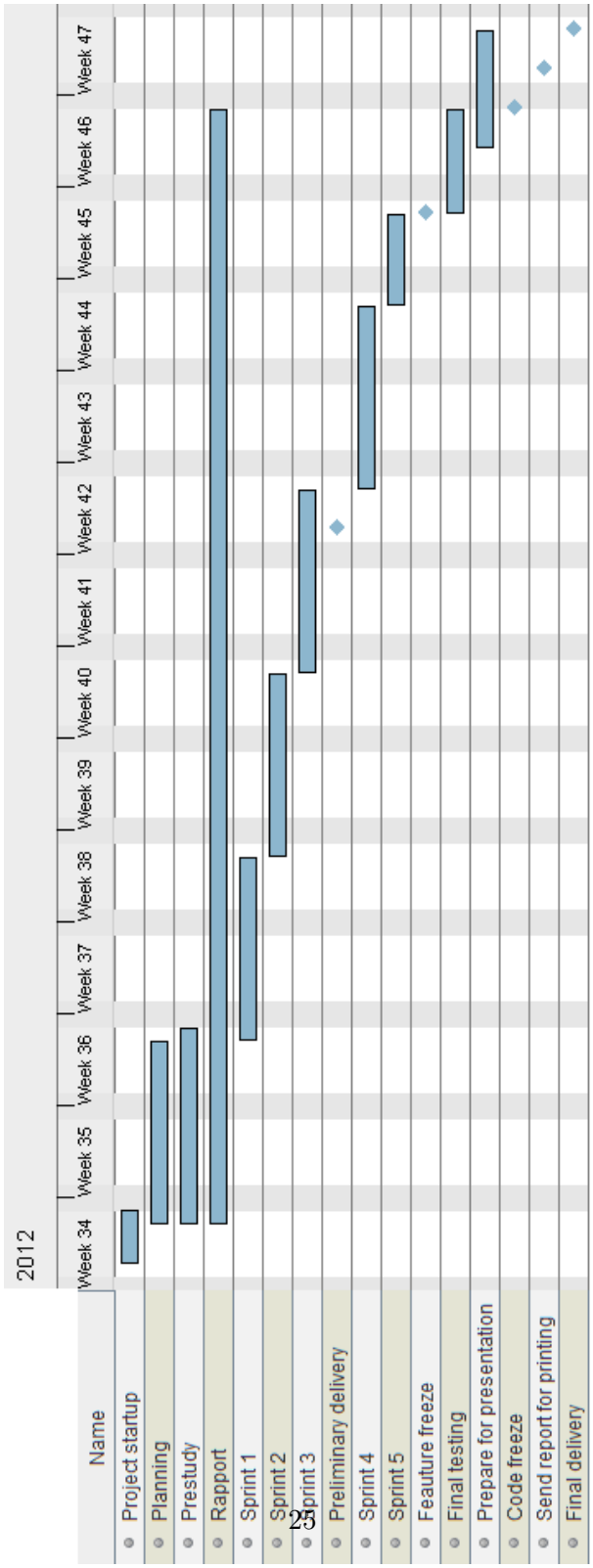


Figure 2.2: The Gantt chart

Feature Freeze: This milestone is scheduled to the 8th of November, and says when we should stop adding new features to our program [69]. By this date, the only changes that could be done to the code are fixing of bugs and refactoring of code. The milestone should be reached by prioritizing all the functionality we want in our system, and start with the highets prioritized tasks.

Code Freeze: At the 16th of November, all the code should be finished [69]. After this date, we are not allowed to make any changes to the code. This milestone should be reached by fixing bugs regularely, by doing tests during the sprints and by using the test phase well.

Send Report for Printing: Our fourth milestone is to send the report for printing. At this time, the report has to be completely finished, and all the text must be free for typing errors. This is scheduled to the 19th of November.

To achieve this, all group members should contribute to the report writing. The project manager should make sure everybody does that, in addition to make sure that the quality of the documentation is good enough.

When all the text is written, each member of the group gets a copy which they have to check for both typing errors and logical errors.

Final Delivery: The final milestone is the presentation of our product, scheduled to the 22nd of November. At this date all of the work has to be finished, and our final product should be as good as possible. This means that it should be thoroughly tested and without errors. It should also satisfy the customer's requirements and be well documented. In addition to having the product and the report finished, we must also have rehearsed on a presentation. All the group members must have an overview of the system and be able to answer questions.

2.2.2 Major Activities

The different bars in Figure 2.2 each represents a task.

Project startup: This is the first task in the chart. It lasts from 21st to 24th of august, and contains the introduction of the task, assigning roles and having the first meeting with our customer.

Planning: The planning task contains the making of requirements, writing of the backlog, making use cases etc. The planning period lasts from 24th of August to 6th of September.

Preliminary studies: The third bar lasts from 24th of August to 7th of September. It contains the time scheduled for prestudy. The prestudy phase contains tasks like

getting to know the technology, gathering information for making decisions about how we should work and getting familiar with the programming environment. However, it is important to notice that at the beginning of each sprint we will include some planning and prestudy if required, in addition to the periods scheduled here.

Report: The report phase is carried out during almost all of the project. Every detail of the project should be documented, and it is therefore important that we write the documentation continuously. The dates for this phase is 24th of August to 16th of November.

The sprints: The main development of the system should take place in five sprints. All of the sprints will last for two weeks, except for the last one, which will only last a single week. Each sprint will start with a meeting with the customer. The first sprint is scheduled to start at 7th of September, and then each sprint will follow straight after the previous one. All of the sprints should be finished by 8th of November.

Final testing: The final testing is done to try to remove all of the errors and bugs in our final product. It will consist of a full system test, in addition to more specific tests. The period will last from 9th to 16th of November.

Prepare for presentation: This task is represented by the final bar of our chart, from 14th to 22nd of November. The scheduled time will be used to make the presentation, rehearse, and make sure that everybody has a full overview of the system.

2.3 Sprint Management

We decided to have a budget of 2 weeks for most sprints. Even though we were not going to work full time on the project, we wanted to only have 2 week sprints. One of the reasons was because we wanted to have feedback from the customer more often. We also felt that with 2 weeks sprints we would be forced to work more continuously, as it did not pass much time between each demo. The only sprint that was not 2 weeks was the last one. This was just 1 week because we wanted to have the final week for more high level testing.

2.3.1 Meetings

The group decided that each new sprint should start after the meeting with the customer. In that way we could decide the user stories and engineering tasks for the sprint from the product backlog together with them, and also discuss some matters if necessary.

2.3.2 Daily Scrum

Tuesdays, Wednesdays, Thursday and some Fridays were the days that the daily Scrum meetings were held, since those were the days all of the team members could meet each

Figure 2.3: Retrospect evaluation board from sprint 1

other. The meetings were the first thing the group started doing in the morning and were mostly used to keep track of progress, get an overview and make sure no one was struggling with their tasks.

2.3.3 Review Meeting

We also had a review meeting with the customer. This was conducted at the customer meetings before we started planning the next sprint.

2.3.4 Retrospect

The retrospect was the written conclusion from each sprint. It also contained the evaluation of the sprint, the results and a burndown chart, showing the work progress visually and making it easy to see if there were some bottlenecks in the sprint.

In the evaluation part every group member was given a set of post-its notes. These were used to write down what we wished to continue doing, what we wished to stop and start doing. This post-its were then put on a blackboard and examined by the group together. Example of such a board is shown in Figure 2.3

2.3.5 Division of User Stories

We chose to divide the user stories into three different priority levels: high, medium and low. A user story with "high" priority was one we absolutely had to finish in the given

time. These were the ones we would start working on in the sprint, and needed to finish during that sprint. A user story with "medium" priority meant that we needed to finish the story during that sprint, but we could move the user story to the next sprint if we failed to complete it during the planned time. User stories with "low" priority were stories that could be done after the ones marked "high" or "medium" are completed.

Each user story had one priority in the product backlog. The ones we were going to work with would also be prioritized for the given sprint. That meant a user story with high overall priority could be prioritized low for the given sprint, as we had other more important tasks in that sprint.

Figure 2.4: Image of us estimating a task using planning poker

After we divided each use story into smaller engineering tasks we wanted to get a good estimation of how big the tasks where. To estimate the story points we used planning poker [70]. Each team member got 6 playing cards A(1), 2, 3, 5, 8 and K(13), representing the magnitude of the task. For each task the team choose one card individually for the estimation, and turned it around when everyone had made a selection. A image of us playing is seen in Figure 3.9. If the presented cards were close in range we knew that we agreed on the difficulty of the task. If there was large gaps in the estimations the one with lowest and highest card stated reasons for their choice, and the team member with the highest knowledge about the subject and task gave their opinions as well. Then we re-estimated. If we felt that the task was bigger than 13 we could put down all cards, we would then divide this task more before estimating.

2.4 Risk Analysis

The risk analysis contains the most relevant risks and problems that we can encounter during the project. By identifying the risks, it allowed the group to be prepared for problems that could arise, make some precautions and strategies for solutions, and in that way be aware of how we could handle the situations.

In the risk analysis we made several tables, shown in Appendix C, presenting the different internal and external risk the group can encounter. For each risk we have identified the activity that is affected, the factor of the risk, how big consequence the risk will have for the project (represented by H: high, M: Medium, L: low), the probability of the risk to happen, the strategy for the solution and the responsible peoples if such a risk should occur.

In this section we will only elaborate some of the risks that we have presented in Appendix C, thus the risks with highest consequence and probability.

Long-term Leave

One of the most likely risks to happen is if some internal or external person in the project has a *long-term leave* due to illness, vacation-trip or other causes. A long-term leave may cause the work to halt, and in the worst case the group may not manage to finish the work planned for a period of time.

The *strategy* for how the group is going to handle a situation if a person is gone for a week or more and is not able to work from home, is that we should organize a meeting where we transfer that persons workload to others, and maybe make changes to the team schedule accordingly. Also, the most important tasks should be prioritized if time gets short. The person *responsible* for detecting and putting the strategy to work is the project manager.

Deadlines Not Reached

Another important and high risk is if the *deadlines* set by the course staff and by the group itself in cooperation with the customer is not reached. By not reaching deadlines, the grade may be negatively affected or the customer can lose trust in the group.

The *strategy* for this risk is to avoid confusions with the deadlines, have clear deadlines that are realistic based on the group's experience and time, and also always have a buffer-time before every deadline for things that may come up last minute. The *persons responsible* for making sure that the group is on schedule and have realistic deadlines are the project manager and scrum master.

Uncompleted Tasks

Another risk that is somewhat tied to the previous one, is the risk of *team members not completing their task*. This can be due to too much work in the project or other subjects, too little time set aside for the task, or the task is too comprehensive for one person. This risk may cause others to take over or help with the task, and in that way increasing the workload on other team members.

The *strategy* here is to maintain a good communication within the group, and in that way be able to detect if some group-members have too much work. The group has also daily scrum-meetings where everyone gets an overview of the status and the work people are doing. In addition all group-members will have time-accounting sheets that are filled out during the project, giving all group-members an overview of how many hours have been used and on what kind of tasks. The *responsible* for this risk is the project manager.

Internal Conflicts

A risk that can lead to serious consequences in the group if it is not handled right away is the risk of *internal conflicts* arising. The consequence of this risk is that the group can experience some tension and dissatisfaction that can contribute to poor work done by the involved parties in a conflict. Also, the customer may think less of the group.

A proposed *strategy* is to maintain a good communication within the group in the form of meetings etc. Here the project manager is the *responsible*.

Changes to the Requirements

The risk of *too little input from the customer*, especially in the early phases of the project or *changes in the requirements* at a late stage in the project may lead to misunderstandings and work halting. In worst case the project can have a risk of heading in the wrong direction.

The *strategy* for this risk is to regularly conduct meetings with the customer and include him in the sprint-planning phases. Also to have a low threshold of when one should contact the customer for questions related to the requirements.

The customer contact has the *responsibility* for this risk.

Loss of Data

A risk with high consequence but rather low probability is the risk of *data loss*. If data gets lost, many hours of work vanishes with it. Without any restoring options the workload may increase dramatically for all members of the group.

An important *strategy* for this risk is to always take backups of everything. The *responsible* for this risk is the project manager.

2.4.1 Internal Risks

Table 2.1: Internal risks

I1	Illness
I2	Long-term leave
I3	Deadline not reached
I4	Overworked team members
I5	Team members not completing their assigned task
I6	Lack of knowledge
I7	Lack of hardware
I8	Team members busy with other subjects
I9	Room reservation
I10	Internal conflict

2.4.2 External Risks

Table 2.2: External risks

E1	Conflict
E2	Requirements change
E3	Lack of input
E4	Tools fail
E5	Loss of data
E6	The customer changes their mind
E7	Not able to reach/contact the customer

Overall tables for the external and internal risks are shown in Table 2.1 and Table 2.2. A full overview of the all the risks and their consequences and probabilities can be found in Appendix C.

2.5 Quality Assurance

This section about quality assurance describes rules and procedures that have to be followed in order for the development process to be tailored to achieve relevant product qualities (e.g. reliability, performance etc.), ensure efficient communication and prevent delays.

Spoken and Written Language

Our group consisted of four students with Norwegian as their native language, and one student from Iran with Persian as her native language. Because of this, we had to use English as our working language. In the beginning, it was a bit difficult to explain everything and find the correct words in English, but after a few days, this was not a problem any more.

This report is written in English, according to the requirements from the course staff. It was also natural for us to write in English, as we spoke English during our work sessions, and it was the only language we all understood. In this way, all in the group could contribute to the report, both with writing, fault detection and correction.

Coding Conventions

We have tried to write clean and easy understandable code, commenting where we found it necessary. The comments are in English, which is the standard to use when coding. We followed a guide on writing Testable Code[46], and wrote the code in a way that made good testing possible. See appendix E for all our coding conventions.

Supervisor Meetings

The meetings with our supervisor were held once a week. Here, we could ask questions about administrative problems, and more general problems about the report. A typical agenda is shown in section B.1.

The meetings with the supervisor were mainly about the administrative in the project, like issues with the group and the writing of the report. The supervisor provided us with useful feedback, as he had more experience from this kind of project work. Also, we summarized the work done the previous week in a written weekly status report. A template for this report is shown in section B.3.

Customer Meetings

The first customer meeting was held at the very first day of the project, the 21st of August. Here we were presented with the task, and the customer came up with some work we should do before the next meeting.

A meeting with the customer was held every second week, in the beginning of each sprint. At these meetings we presented the results from the previous sprint and plans for the next sprint. The customer came with reviews and additions to what we presented, and together we planned the next sprint. A typical agenda is shown in section B.2. These meetings were held via Skype.

Chapter 3

Preliminary Studies

This chapter contains the results from our preliminary studies phase. The most important information we found about the subjects we studied are written in this chapter. Also, we have included some discussions about both the technologies, methodology and the reasons for the choices we made.

In addition, we have looked at existing products that have similarities with the product we should develop. We have written about different testing methods and we have taken a look at the tools we are going to use, both for development, collaboration, report writing and other tasks.

3.1 Development Methodology

For the development methodology we chose to looked closer into Waterfall and Scrum. They are both used in many developing projects around the world. We have learned about them in previous courses, and we had experience using them from before. We had many other aspects that was new for the team, so we wanted to use a methodology we already where familiar with.

3.1.1 Scrum

Scrum is an agile development method which in recent years have become highly popular[15]. It is an iterative process where an iteration is called a sprint. The iterations is shown in Figure 3.1. Typically one sprint last 2-4 weeks[9]. A team working with Scrum is divided into different roles[48]. The scrum master should help the team to work as effective as possible. The product owner is the person representing the customer. He should ensure that it is still valuable for the customer to continue the development.

In the initial phase a product backlog is made[1]. All functionality of the product should be described here, but this list could be changed along the way. Before each sprint some of

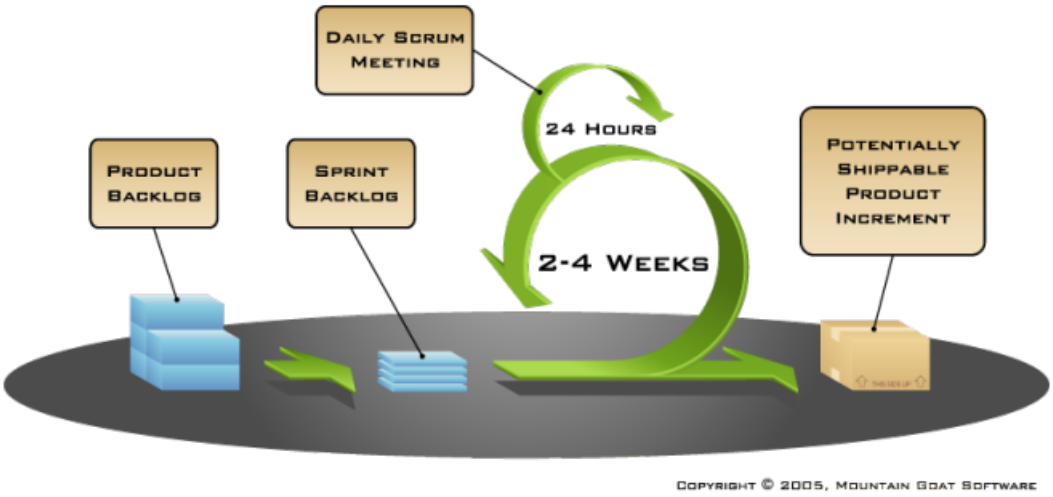


Figure 3.1: The basic workflow in Scrum development [26].

the items in the product backlog is chosen for the sprint backlog. This is the functionality that will be implemented during the next sprint.

At the end of each sprint the product is demonstrated on the sprint review meeting. The product owner and other interested stakeholders should be at the meeting and give their feedback. It could also be useful to have a sprint retrospective after the sprint review. In the retrospective each team member should give their opinions on what they liked in the previous sprint(continue doing), what they did not like(stop doing) and what the team should start doing[50].

Flexibility is the main advantage of using Scrum[49]. The implementation of one part of the system can start before all requirements and design is finished. This is good for the customer because they do not need to know all the requirements before the start, and they can see the working system after each sprint, and this could lead to changes to the product backlog. The trust between the customer and the developers will generally increase, and a positive culture is created in which everyone expects the projet to succeed. Using Scrum also has disadvantages, like lack of documentation, making it harder to change team members along the way, or making it harder for another team to maintain the system.

3.1.2 Waterfall

The Waterfall method shown in Figure 3.2 is a sequential design process, which has been used for a long time[9]. The process is divided into seven phases: requirements specification, design, construction, integration, testing, installation and maintenance[28], and the documentation is produced in each phases. When one phase is finished the next one will begin. One should avoid going back in phases as much as possible, thus radically changing

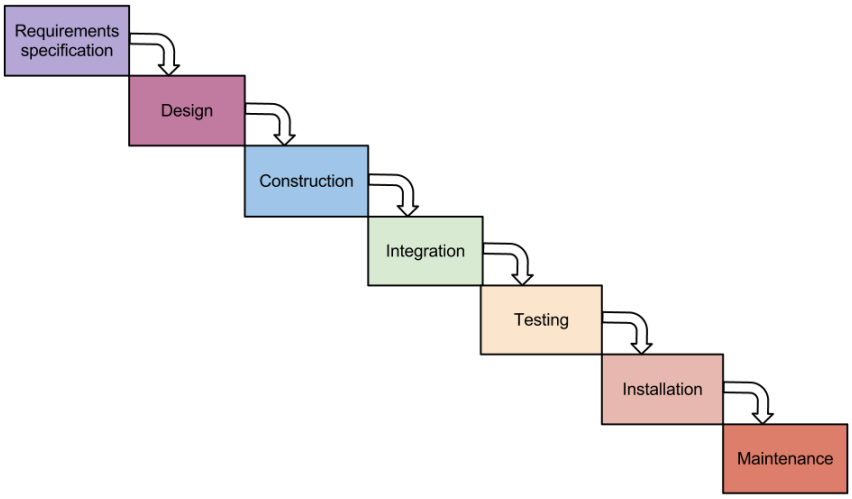


Figure 3.2: The waterfall model[27].

requirements would be very costly when using waterfall.

One of the advantages of using the Waterfall method is that you can find and correct a lot of flaws early in the design phase. However, problems can arise if something that looked simple becomes hard to implement. Going back some phases and redesign can be expensive and create mismatches between the documentation and the implementation. Due to the inflexibility of the Waterfall method it is important that the customer knows all the requirements at the start of the project.

3.1.3 Criteria for Methodology Choice

Here we represent the criteria we used when making the choice of which methodology to use. The criteria and their importance is shown in Table 3.1.

We wanted high flexibility in changing requirements, since it should be easy for us to make changes to the requirements during a process if we or the customer desired to do so. The importance of this criterium is medium.

It is important that it is easy for the customer to give us feedback on a regular basis, so that we know that he is pleased with the system, and that we can find possible misinterpretation as early as possible. The importance of this criterium is high.

We wanted to produce a lot of documentation, showing how we worked, what kind of research we had done and how our architecture looked. Since the documentation is one of the most important criteria the sensor will use for the grading of our project, the importance of this criterium is high.

We wanted a methodology that we would be the most relevant for us to learn more

about. Considering what we have worked with before and what we most likely will work with in the future.

We did not want to do all the planning in the beginning as this is the phase where we know the least about the system. At the beginning we needed to learn to program, learn three new frameworks and all that in C#, a programming language just one of us had used before. This would be very hard to plan and estimate in a good way right away. We would know more about the system when we had work more with it a bit, and then also be able to make better plans accordingly.

We also wanted the cost of redesign after the beginning phase to be low, as we might do some flaws in the first weeks, because of inexperience with the programming language, frameworks and architecture design.

Table 3.1: Criteria for the methodology

Criterium	**Importance**	**Scrum**	**Waterfall**
Documentation	Very high	Little	A lot
Ease of customer feedback	High	Good	Bad
Cost of redesign after already started phases	High	Can be cheap	Expensive
Flexibility in changing requirements	Medium	High	Low
Needs all the planning done in beginning	Medium	No	Yes
Relevance of learning	Medium	High	Medium

3.1.4 Choice of Methodology

Before making the decision we wanted to have an overview of which criteria the two different methodologies fulfilled.

Scrum is an iterative process, thus having high flexibility in handling changes to the requirements. Requirements could be changed, removed or added between each sprint, and it should not create big problems. The customer gets to see a runnable system at each sprint review meeting, making it easy for them to give informative feedback. Using Scrum, documentation is normally very limited. As it is normal to have regular contact with the customer, documentation is not needed in the same way as it is with waterfall. We though it would be highly relevant for us to learn more about Scrum as it is popular in many consultant firms and their customers. We don't need to do all the planning in the beginning with Scrum, because we have a planning meeting each sprint. The cost of redesigning after the beginning phase should not be too high, because it is iterative you do

not have to change back to the design phase in order to do a redesign.

Waterfall is a sequential process, a change in requirements would mean that the whole project had to go back to the requirements specification phase. In other words the flexibility is low. It could be hard to get good customer feedback in the beginning stages as the customer only has the documentation to look at, and no runnable system. It would also take more time for the customer to get through it, and might require a very technical customer for them to understand it. For our learning Waterfall is relevant, but not as relevant as Scrum. All the team members had worked with waterfall in previous projects. Waterfall requires that all planning should be done in the beginning. Having to redesign the system after the beginning phases would most likely be very expensive as we would need to go back to the design phase and start going through the phases one more time.

Looking through the Table 3.1 summarizing the criterias and how the methodologies fits, we can observe that Scrum is the best methodology for us to use in two out of three of the criteria with high or better importance for us. Using only this information the decision is still not obvious, as documentation is of very high importance this project. When considering the medium important criterias we can see that Scrum comes out as the best for us in all three criteria. At this point we can finally decide that Scrum is the methodology we are going to use.

We are going to look into ways to make enough documentation even though we are using Scrum. The customer aslo wanted us to use Scrum. He wanted Scrum because Itera Consulting has adopted this methodology internally for all the projects in the last year, and he wanted us to have the same approach to the project. Their reasons for adapting Scrum were that it improves communication between team and customer, it provides an open forum, where everyone knows who is responsible for which working item, it can increase team and project efficiency, and problems are more transparent and easily solved.

3.2 Existing Products

In this section we will look at existing products that are similar to the technologies we will use and that aims to obtain more or less the same functionality as our project. Currently there are some systems that are very similar to the one we will make. Some of the systems have more or less all the functionality as ours will have, but does not cover the same usage area, such as the Samsung smart TV or any motion controlled game (Kinect game for the Xbox)

3.2.1 Samsung Smart TV

The Samsung smart TV features voice, gesture and face recognition[56]. A picture of the product can be seen in Figure 3.3. The TV supports controlling and adjusting of settings by hand waving and by voice. The television uses a HD camera for the motion controlling. It can also show 3D contents, when using a pair of active-shutter glasses.

Figure 3.3: Samsung Smart TV with voice and gesture recognition[56]

The Samsung TV system is designed for "entertainment" use, so one can only use simple hand gestures for movements (like scrolling down a page or switching channels), and the voice controlling is restricted to some set of commands. The Samsung smart TV has many of the same features that our system has, but contrary to their system, where only some parts are motion and voice controlled, ours will be fully controlled by the use of motion and voice.

3.2.2 Siri Personal Assistant

The Siri is an intelligent personal assistant which works as an application for Apple's iOS[57]. The application can be seen in Figure 3.4. Siri uses a natural language user interface for voice controlling. Siri can answer questions, make recommendations and perform some actions (like web searches and making new entries in the calendar). Our system will have voice controlling, but on a different level than the Siri, as our voice controlling will be restricted to a set of predefined commands.

3.2.3 Xbox Games with Kinect

"Kinect sports" is one of several Xbox games that use motion controlling (with Kinect) in the game. It utilises the Kinect motion sensing peripheral[58], and is a collection of different sports simulations designed mainly for the demonstration of the Kinect. The players stand in front of the Kinect sensor and mimic actions performed in real-life, e.g. jumping and swinging the arm to smash a volleyball over the net, like in Figure 3.5.

Figure 3.4: Apple's Siri Personal Assistant application with voice recognition[57]

Figure 3.5: Kinect sport, an Xbox game with voice and motion recognition[59]

3.3 Testing

This section gives an overview of the most common types of testing. Each type falls in the scope of either black-, gray- or white-box testing, which are explained below. We also describe Test Driven Development, and how we did testing in our project.

3.3.1 Types of Testing

Black-Box Testing

When dealing with black-box testing you only test the functionality of an application, and not the internal structure, thus the name black-box[16]. The testing method can be applied to all levels of software testing, from unit, integration, to system and acceptance testing. In general black-box testing covers testing of the most "higher levels".

With black-box testing the tester should normally not need any knowledge about the code or structure of the application, or any programming knowledge in general. The testing person needs only information about what the system/application is supposed to do, not how. Thus, he should only input something and see if the expected output is produced.

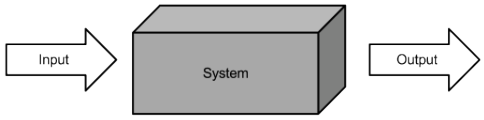


Figure 3.6: Illustration of black-box testing[16]

White-Box Testing

Unlike black-box testing that tests an applications functionality, white-box testing covers the internal structures[18]. To design test cases for white-box testing the tester should have programming skills and an internal perspective of the system. The idea is that the tester should choose inputs for the parts of the system he wants to test, he then "traverses" paths through the code with the input and determines the appropriate outputs. White-box testing is usually done at unit level, but can also be conducted when performing integration or system tests.

Gray-Box Testing

Gray-box testing is a combination of white- and black-box testing[17]. It uses the straightforward techniques of black-box testing and the code targeting in white-box testing. The testers have some information about the system they are testing, but do not need to have any information or knowledge about what happens internal in the methods.

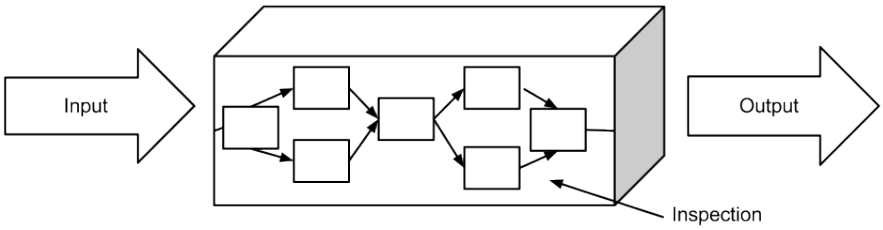


Figure 3.7: Illustration of white box testing[60]

Unit Level Test

A unit is the smallest testable part and it is tested independently of other units[20]. It is important that the developer understands the specification of the system, because testing happens continuously as the units are developed.

Unit testing is white-box testing because we are testing the code while implementing it.

Integration Test

The integration level tests are tests for the interfaces between the different sub-systems of the system, or on individual component basis[61]. Integration testing seeks to verify that the interfaces works properly, and gives the most correct data values in all possible situations. Integration testing is also used to verify the integrity of the entire system, to confirm that all interfaces are working together.

The integration testing is done after the unit tests. A sub-system is tested one-by-one to ensure that it can work independently before more sub-systems are "hooked" up to ensure that no subsystem interferes with another sub-system in a manner that is not expected.

For the test to be completed, all subsystems must be able to act independently of other subsystems. In addition, each subsystem must be able to function while other subsystems are running without significant performance or reliability loss.

System Test

System level tests are to be conducted on the system as a whole and they should be performed at the end[62]. The testing is executed on a complete and integrated system to see how it all works together.

The testing on system level is primarily black-box testing, where the entire system will be tested at once, but where smaller subsystems will be in focus.

Since the tests in this level are dependent on all sub-systems and components working independently, these tests must be passed before the system level tests can be completed.

The system level tests are completed when the tests shows adequate performance and reliability in a number of environments, with no single sub-system failing. System testing verifies a system by checking it against the published user or system requirements.

Functional Testing

Functional testing is like system testing - a black-box test[63]. Functions are tested by feeding them a specific input and control if they deliver the expected output. Functional testing verifies a system by checking its behaviour against predefined specifications and/or design documents.

Usability Testing

Usability is defined by ISO 9241-11 as: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"[23]. Meaning that during user tests we should observe if the user is able to perform the intended task, that the user does not use too much time, and that the user is satisfied.

The usability tests are to check if the system and its user interface is understandable for users. This tests are carried out throughout the project, where you get feedback from the customer and make corresponding changes if necessary.

Since usability is of importance in our project, we will have usability tests, but with some modification from the usual practice, as we could not make any paper-prototype of our system in an early phase of the project, so the testing could only be done near the end when most of the systems functionality was implemented.

3.3.2 Test Driven Development (TDD)

TDD is a programming method where you are doing testing iterations[11]. The iterations start with making or rewriting a unit test until you have a test that fails. Then you implement the code to make the test run right. Finally you can make internal changes to the code, like re-factoring, as long as all the test works after these changes. Iterations can continue as long as the team wants or until we have a finished product. The entire process is displayed in Figure 3.8.

TDD is good to use for us because it forces us to make clean and testable code, work on a single task, divide the program in a good way, and will lead to better test coverage.

3.3.3 Testing in Our Project

For this project, it is very important to choose testing techniques that helps us to make good and understandable code, as we are multiple programmers. Also, as we are using Scrum as the methodology, we will have many iterations, as a result of that many changes

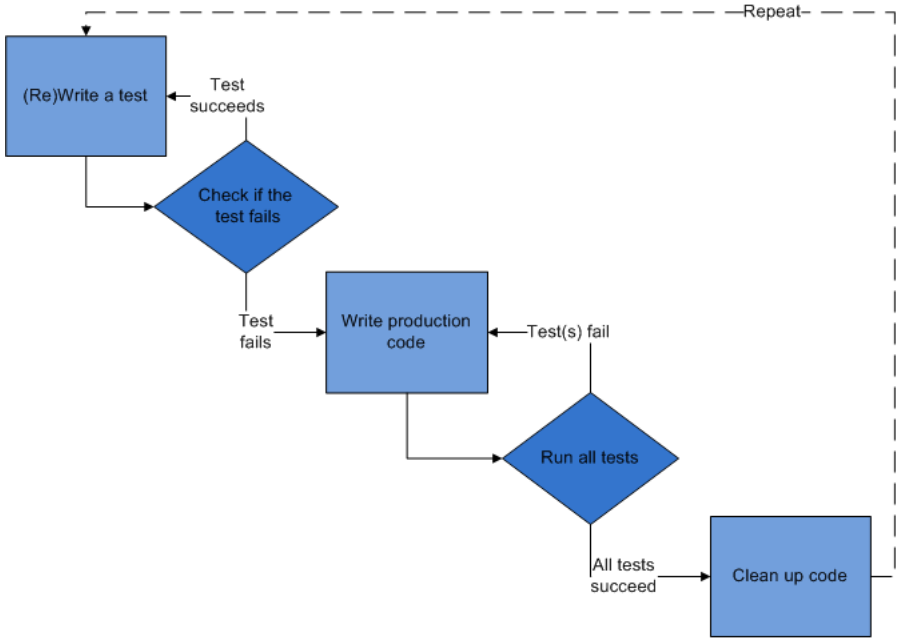


Figure 3.8: Test-driven development[29]

will be made along the way. This means that we will make many internal changes to the code, like re-factoring, so testing regularly along the project progresses is important and will save us a lot of time, compared to do all the testing at the end.

Our customer is not in Trondheim. Therefore it was also important for him to know that everything was working as it should and that things had been tested along the way. The customer had also a desire for us to use the test driven development approach in our project, mainly because it is a good and secure way of writing code.

It was very clear from our needs and the customer's wishes, stated above, that test driven development was the programming method we should choose. As TDD is a method that you do testing iteration with making of unit test, we agreed upon that unit testing was something we were going to do.

The goal of unit testing is to isolate parts of the system and show that they are correct, so it was very important that each unit-test is bug-free. The unit tests were therefore created before the writing of the units. If the test passed, the code was considered complete.

The advantage of unit testing is that it finds problems in an early stage. The tests need to cover all the code, in the other words, the units must meet all the requirements of the tests.

As there were three more or less "stand-alone" parts of the system: the Kinect, the microphone and the model, it was needed to verify that these components were working

alone and together in the way we wished them to do. We needed to make tests to confirm that the Kinect was able to detect motion and transfer this to the model, that the microphone detected and recognized voice as it should and transferred the commands into actions on to the model. For these needs here it was clear that some types of integration testing needed to be performed. First to test the components as stand-alone units and then in accordance with the rest.

System test is something that will be performed at the end of the project, verifying that everything is as it should be. The reason for choosing system testing is that it is the best test method when wanting to test the whole system and all its functionality.

We would also perform *some* performance tests to make sure the performance demands from the customer was met. But the performance part was mainly covered and assured to be good in the system in the chosen tactics. The tactics can be found in section 5.2.

At the end some form of usability and user acceptance testing was to be conducted. User tests to see how user-friendly the system was and acceptance test on the final product where it determined if it is approved or not.

To read more about the testing done and how we did it, see chapter 12.

3.4 Tools and Libraries

This section gives descriptions about which tools we used and discussions about why we used them instead of others. The tools could either be hardware components, other programs or frameworks.

3.4.1 Kinect

Kinect is a motion detector made by Microsoft, which can be seen inFigure 3.9. It is a hardware device that contains two cameras and one infrared projector[64]. The first camera is used to capture RGB video, the second for infrared light. The main use of the device is games on Xbox 360, but it could also be used for small applications on Windows. It was made by Microsoft in order to broaden the appeal of their Xbox console by making it possible to play games without the traditional game pads controllers. Kinect sold 8 million units the first 2 months[65].

Another device similar to Kinect is Leap[66]. It can recognize any finger motion and the price will be much cheaper, but it will not have voice recognition and it will release in the beginning of 2013. Accordingly, another reason to use the Kinect is the lack of other similar alternatives. As a result we decided to use the Kinect.

Our customer wanted us to use Kinect. They wanted us to use Kinect for detect the hand motion to rotate the air traffic space and redirect the position of planes using its voice recognition. They had the experience of working with Kinect in a static environment. There the user threw imaginary paint on the screen where the T-shirt was in order to make

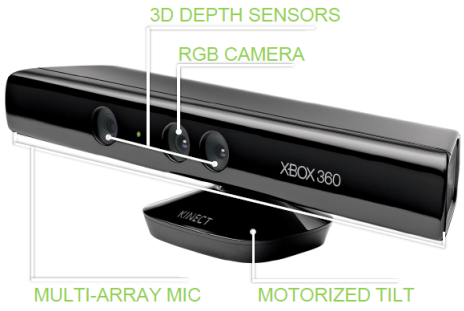


Figure 3.9: Image of the Kinect and the different sensors, captured from a video by Microsoft[54]

a pattern on the shirt. Then, the image was sent for printing on a real T-shirt. Now they want to check the Kinect usage in a dynamic environment.

Kinect is relatively cheap to get, and the customer will provide us the device to work with, it should fit well with the other Microsoft solutions we are going to use, and should be fun to learn and work with. It works by projecting infrared light on everything that is in front of it. It then calculates the position and distance between the light source and the corresponding point of reflection. The device also has an array of built in microphones in order to capture the input voice and a motor to capture motion across a view. It also has some useful examples to start working with and supportive forums and websites to answer the questions that may arise during the implementing of the project.

Here are more details on the device[67]. The horizontal field of view of the camera is 57 degrees and the vertical one is 43 degrees. The depth sensor can track from 1.2 to 3.5 meter and the device can with a built in motor tilt up and down 27 degrees. Its camera provides 3D information about each frame which is useful for image processing algorithms. For this purpose the RGB camera and infrared sensors are working together. For the RGB camera there are 2 options. It can be a 640 x 480 pixels image stream at 30 frames or 15 frames per seconds[40], or a 1280 x 1024 pixels image stream at 15 frames per second. The depth sensor provides the resolution 320x240 pixels with a 16 bit depth value associated with each of the pixels. The depth sensor chapters light in 3D array via monochrome CMOS (Complementary metaloxidesemiconductor) chip.

3.4.2 Programming Language

In this section we discuss the different programming languages we considered, and what we ended up with.

Java

Java is one of the most used programming languages in the world[4]. Its syntax is derived from C and C++, but unlike them Java has automatic routines for a lot of lower level facilities, such as memory management[71]. Normally Java is compiled to byte code which can be interpreted by any Java Virtual Machine regardless of the underlying computer architecture. This approach makes Java ideal for cross-platform development.

C++

Another very popular programming language is C++. It has manual memory-management and is faster than the other programming languages we considered[5]. One of C++'s application domains is video games, which has influenced the creation of many 3D libraries. C++ is also supported by the Official Kinect SDK, a software development kit for motion sensing.

C#

C# is a programming language developed by Microsoft[72]. It is syntactically very similar to Java, but it implements the .NET framework. Like Java it has automatic routines memory management, and other lower level facilities. C# is also supported by the Official Kinect SDK. Both Java and C++ are better than C# when it comes to performance.

Criteria

When we were choosing which programming language to use, we had a few important criteria with different level of importance. The criteria can be seen in Table 3.2.

Table 3.2: Criteria for programming language

Criterium	**Level of importance**
The language must support motion detection frameworks	High
The language must support speech recognition frameworks	High
The language must have support for 3D development	High
The language must have good performance when rendering 3D	Medium
Group members have previous experience with the language	Medium
Easy to work with	Medium

Decision

We decided to use C# as our programming language, although most of the students in our group had previous experience with Java. This project did however present us with some challenges which we had never faced before, namely 3D modeling, motion detection and speech recognition. Java supports some 3D libraries like OpenGL, but Java's performance is rather poor when it comes to 3D rendering. The range of motion detection and speech recognition frameworks are also rather poor for Java, so we decided to use another programming language for our project.

Only one of the group members had programmed in C++ before this project. C++ is considered to be a hard language to work with, mainly because of pointers. It is however one of the most powerful languages in terms of processing efficiency, and is supported by speech- and motion detection frameworks, and 3D libraries. Even though C++ has a wide variety of frameworks and development kits suited for our needs, we felt the difficulty of adjusting to C++ would outweigh these benefits.

The last language we considered was C#. It is syntactically similar to Java, and even though only one person in the group had previous experience with it, we felt that choosing this language would pose no problem. Since C# is a .NET language, it supports a multitude of Microsoft frameworks, among them Microsoft.Kinect and Microsoft.Speech. It is also used for game development for Xbox and Windows Phone, which means it has a lot of 3D libraries at its disposal. The combination of easy transition, support for multiple frameworks, and that our customer wanted us to use C#, made us end up with this programming language.

3.4.3 Frameworks Used in the Project

In this section we present the different frameworks used. The section addresses the tools we managed in order to use the different frameworks.

Motion Detection

As discussed before we are going to use the Kinect for motion detection part of our project. In order to use the Kinect we need a program called Kinect Studios that installs the plug-ins we need implement the Kinect using Visual Studio. This is done using the .NET framework.

Speech Detection

The Microsoft Speech Framework makes speech recognition applications for Windows, and is made for use with the .NET framework. It has speech recognition engines in multiple languages, including English and Norwegian. It also contains examples of how to do some of the development. This will probably be very helpful for us. We can use the Microsoft

Speech Framework with the Kinect. At this point Norwegian is not supported by the Kinect, but it will be in the near future[30]. We are going to use the Microsoft Speech Framework because it is compatible with the Kinect and the XNA Framework. We wrote more about speech detection in subsection 3.5.6.

Framework for 3D Applications

There are multiple possibilities for a 3D application. Here are our criteria for choosing the one that fits our need the best. We needed a framework that would easily support the connection with Kinect. The customer wanted a program with good performance, so we needed a framework that could give us at least medium performance. As developers that haven't work much with 3D applications before, we needed a framework that was easy to learn. In order to not use too much time in the beginning it was important for us that there were many examples and tutorials available. As programmers we also work better with a framework that we want to learn more about, as this will increase our motivation and most likely give the best result.

The Microsoft XNA Framework 4.0 (XNA) is a framework made by Microsoft, that is easy connected to the Kinect. Microsoft wanted a better system for creating small games easier for their Xbox 360 game console. They wanted to make the platform for downloadable games bigger.

XNA can handle both 2D, 3D, various input devices and audio. Using the XNA Framework makes some of the programming easier. It is easy to create the project files, and to get the program running from the start. It should also be easy to learn.

It has a sets of useful tools. One of them is the runtime environment manager, that does task like garbage collection automatically. Even though it makes things easier there are some drawbacks, such as keeping objects alive for too long. This would lead to a drop in the performance. For XNA there is enough tutorials and examples to get us starting. The programmers were highly motivated to learn XNA, because of other interest like game making.

Another option we looked into was the Windows Presentation Foundation. WPF is a framework for making windows based client presentations, that is easy connected to the Kinect. The core of WPF is vector-based rendering. Vector based graphics looks good on all kind of screens independent of resolution, because of its ability to scale. WPF includes XAML, data bindings, 2D and 3D graphics, controls and layouts, animations, styles, documents, text and typography. WPF is a part of Visual Studio .NET and it can use all .NET libraries. Using WPF would give us more options for tweaking the graphic manager and may contribute to performance, but that would take some time figuring out. On the more basic areas WPF have enough tutorials available for us. WPF will be more difficult to learn because you need to create more XML files to get the 3D environment up, and we have not worked much with this before. Also, we have not heard much about WPF before this project, so we did not have any special motivation for working with WPF.

Table 3.3: 3D framework options and how well they fitted our criteria

Criterium	**XNA**	**WPF**
Performance	Medium	Medium to high
Learning	Easy	Harder
Many tutorials and examples available	Yes	Yes
Motivation to learn	High	None special
Possibility to connect with Kinect	Good	Good

We consider all this criteria as very important when making the choice. From Table 3.3 we can see that XNA should be more easy to learn and that we are more motivated to learn about it. We made the choice of using XNA.

3.4.4 Mocking Unit Test

For mocking unit test in C# we found a testing framework we could use. This is called Moq[75]. In order to use that we just had to add its moq.dll to the project as a resource. This could be useful if we needed to use mocking to simulate dependencies when unit testing a single class or method.

3.4.5 Version Control

This section contains information about different version control systems we considered and our choice for this project.

A version control (often called revision control or source control) system is a repository for files which is monitored for changes. If a file is changed, the change is tracked along with who changed the file, why the file was changed, etc. In this section we will discuss why we needed a version control system, which one we chose, and why we chose it.

Apache Subversion

Apache Subversion, which is usually abbreviated SVN, is a centralized version and revision control system[81]. It was released in 2000 and has roots back to the Concurrent Version System (CVS). The main purpose of Apache Subversion is to allow users to be able to restore their data to previous versions, as well as easily synchronize their work with others working on the same project. Apache Subversion is free to use, and is widely used by developers around the world.

Table 3.4: Criteria when choosing version control system

Criterium	**Level of importance**
Easy to use	High
Works well with the other tools we are using	High
Previous experience	Medium

Git

Git is another version control system. It was originally created by Linus Thorvalds and released in 2005[79]. Since its release it has been adopted by many. Unlike Apache Subversion, Git is a distributed version and revision control system[3]. This means that users have a complete copy of the repository when working. If a centralized repository is lost due to system malfunction or other reasons, it can easily be restored with only small losses, if any.

Team Foundation Server

Unlike Git and Apache Subversion, which are version and revision control systems, Team Foundation Server is a full software development and management system[80]. Team Foundation Server, which is usually abbreviated TFS, contains systems like version control, revision control, bug reporting and requirement tracking. Team Foundation Server is developed by Microsoft and is the only product we considered which costs money.

Criteria

Our group had many reasons for using a version control system. We are 5 people working on a single project, so we were bound to be working on the same files from time to time. A version control system offers solutions, such as file merging, to this problem. We can also tag why we have edited certain parts of a file, so that others can easily understand changes in our code base.

When we compared the different version control systems to each other, we had certain criteria that mattered more than others. All our criteria are listed in Table 3.4

Our Choice

Neither of the members of our project had any previous experience with Apache Subversion. It is easy to use, and have plugins for Visual Studio, which is our development environment

of choice. Since it completes the same purpose as Git, which two of us were familiar with, we decided not to use Apache Subversion at all.

As earlier mentioned, two of our members had previous experience with Git. It has a lot of tools we could use in combination with Visual Studio, and has the advantage of working just as good when the developer is offline. Unfortunately Git has a steep learning curve, and it takes time to master, which made us reluctant to use it.

Our customer requested that we use Team Foundation Server, which he would supply and set up for us. This would remove the negative effect of Team Foundation Server costing money, which is a large minus for students like us. Team Foundation Server has a low threshold to start using, and comes with great integration for Visual Studio. Since it also has a couple of extra tools, specifically for scrum which we use as our development method, we decided to use Team Foundation Server. Unfortunately the server was delayed, so in the meantime we used git as an intermediate solution.

3.4.6 Extra Tools Used in the Project

GanttProject

For creating our Gantt chart, we have used a free tool called GanttProject[31]. This is an open source program for project scheduling and management. The software can be downloaded from[31], and is free and without any registration needed.

For our project we needed a tool for the presentation of the Gantt chart.We could have used an Excel-sheet, but that would require a lot of manual work. By choosing the GanttProject program we could easily input the activities and their time consumption, the program then made an nice output of a Gantt-chart. Another reason for choosing the tool was that it was free.

Violet UML Editor

To create the UML diagrams, we have used a free tool called violet UML[32]. This tool provides the user with a simple graphical interface, where the most of the interaction is based on drag and drop. The tool can be downloaded from[32]. We used the editor for making use case diagrams, package diagrams, sequence diagrams and class diagrams.

Violet UML was used because some of us had used it before in the "Common project" in the 2nd year at NTNU. They were satisfied with it, and we decided therefore not to use any time searching for an alternative. The other main reason was that it is compatible with both Windows and mac, so everybody could use it at their computer.

Google Docs

Google docs is a way to share documents between multiple users, making it possible for more than one person to edit the document at the same time[33]. It is also possible to create collections and share them. The documents are stored at an online server, making them available as long as the user has an Internet connection.

We have used Google docs to structure all of our documentation and to use basic spell check, before adding it to LaTeX. We created a collection named "Customer driven project," which again consists of collections for all the different chapters in our final report, in addition to a chapter with meeting agendas and minutes. We chose Google docs because of the ability to collaborate at the same time on documents, everyone could have easy access to the documents and spell checking.

LaTeX

LaTeX is a free typesetting software used for documentation. LaTeX is a simple tool, with easy-to-learn formatting commands, making it a easy to learn and use. The idea of LaTeX is to "help" the writers to focus on the content of what their are writing about and not the visual presentation (since LaTeX takes care of that).

We used LaTeX in our project because it is widely used in academia, it provides an easy way to make cross-references in the document, insert tables and figures, and "automatic" numbering and layout of the document.

Blender

Blender is an open source 3D computer software product[35]. The program can be used to numerous 3D tasks, like 3D modeling and texturing, game creation, making an emulator, 3D applications and animation. It is considered as one of the best of the free programs for 3D modeling. The program was downloaded from bleeders homepage[36].

We decided to use Blender because we needed to make 3D models with textures. As we were going to make an airplane model and a model of the ground, and maybe more if we had time left in the later stages. Blender can export the 3D model to a .fbx file. This was the most important criteria for us when deciding to use this program, because XNA can import a .fbx with textures, we could then use it without many other steps. The second most important criteria was that it was free. We also considered 3D max, that also can export to .fbx, but it had only a 30 day free trial, and we needed more than that as the project expands over several months.

Visual Studio 2010

Visual Studio 2010 is made by Microsoft and is an integrated development environment. Is supports all of the languages that implements .NET. In addition it is also possible to download and integrate other framework packages.

In our project, we decided to write the code in C#. Since this is a .NET language, it was naturally for us to use Visual Studio as our develop environment. Visual Studio is a very expensive program to buy, from 4000kr for the professional edition, but fortunately all of the NTNU students under ID receive a free license to it.

The XNA is a framework which introduces methods and classes to work with 3D objects. To get access to this framework, all we had to do was to download two packages from Microsoft and install them on our computer. Then we could use all of this in Visual Studio. The two packages were downloaded from[37] and[38].

Skype

Skype is a tool that makes it possible to have video conferences over the network connection[41]. Each participant in the video conference must download Skype to his own computer. The standard version of Skype is free, but it is possible to upgrade to the premium version for a small amount of money. As long as you are going to only have one to one conversations, there is no need for the premium version. However, since we were five persons in our group, and the customer representatives were two persons, we needed the feature the premium version had. Fortunately the customer had Skype premium, so we could conduct video conferences with everybody at the same time, as long as the customer was the one initializing the conversation.

The reason for using Skype was that since the customer had his office in Oslo and we were in Trondheim, it was much easier to have video conferences instead of him traveling to Trondheim, or the group traveling to Oslo every second week. Also we had no experience with other video-conference tools except some of us that had used google's video conference, our customer preferred to use Skype as this was the tool used in their company. The threshold for using Skype was also pretty low, because everybody had used it before.

Facebook

Facebook is a free social network[42]. It is accessed from www.facebook.com, and each user creates his own personal account. The way we used Facebook was that we made a private group where only the participants in our group had access. At this group we posted rooms for the next meeting and links to websites that might be of interest to the rest of the group. Also, the tool was used to send short messages to the whole group, to let them know if you would show up late etc.

We used Facebook because this is the one site (in addition to the mail account) each of the group members checks multiple times a day. It is also easy to comment on things posted, and there is a feature that says how many of the group members that has seen a message. So one can make sure that everybody has received all of the information.

Teamviewer

Teamviewer is a free program that gives a user access to another machine[43]. In this way, it is possible to sit in different locations and see the same screen. In our project this tool was essential when we had our meetings with the customer. By using the teamviewer we could demonstrate our product for them at the sprint review meeting. This was important because the customer works from Oslo, so we need to have meetings online.

Gimp

Gimp is an image editing program[45]. We were going to use Gimp because it is much more advanced than Microsoft Paint, included in Windows. An example is the option of having layers. Gimp is free to use. We used Gimp to make some of the figures in the prestudy chapter. We also edited the colors and added arrows to the photos in the Kinect motion research chapter.

3.5 Development Technology

This section contains the result from the research on development technologies. It was need to do research into 3D calculations, motion detection and speech detection before we could develop those parts of the system.

3.5.1 3D Space

The 3D space consists of the x, y, and z-axis, thus a position in the 3D environment consists of a X, Y and Z position. The Figure 3.10 shows how the XNA 3D coordinate axis system looks.

In order to rotate a 3D model there are three rotation one can use. These are called Roll, Yaw and Pitch[47]. From the figure Figure 3.11 one can see how they will affect the plane. The yaw rotation will affect the horizontal direction the plane is going in. It is changed by rotating the plane around the y-axis in the direction shown in the figure. Pitch will affect the vertical direction, a positive pitch means that the plain is rising. Pitch is changed by rotating the plane around the z-axis. Roll is changed by rotating the plane

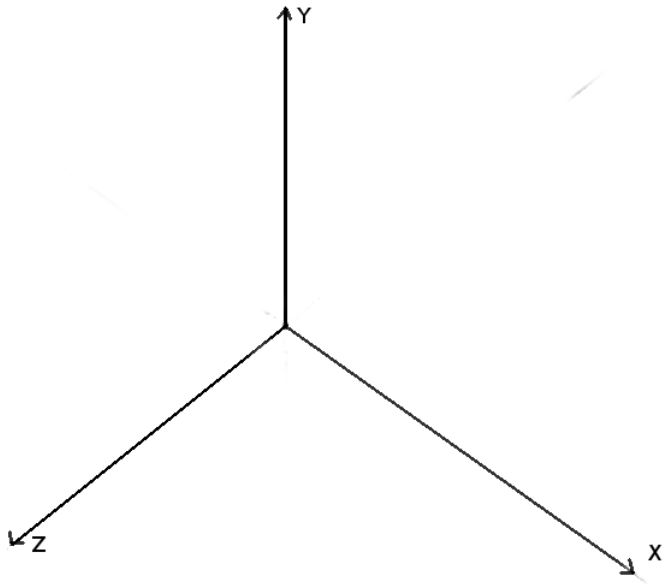


Figure 3.10: 3D space and axis

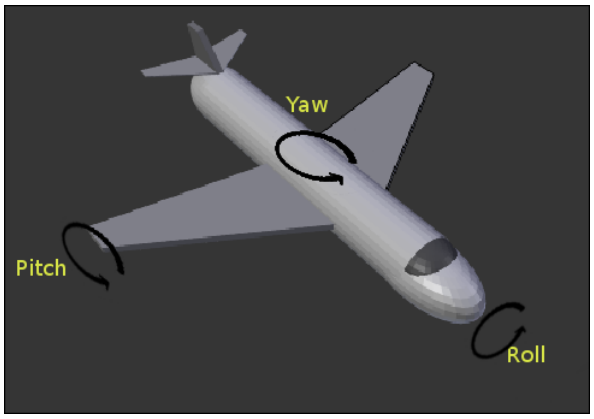


Figure 3.11: 3D rotations made from 3D flight Dynamics figure[47]

around the x-axis. Using roll rotation is not needed for our prototype, but we will make it easy to add that, if that is needed in the future. There are methods for rotating the 3D model around all the three axis in the XNA framework.

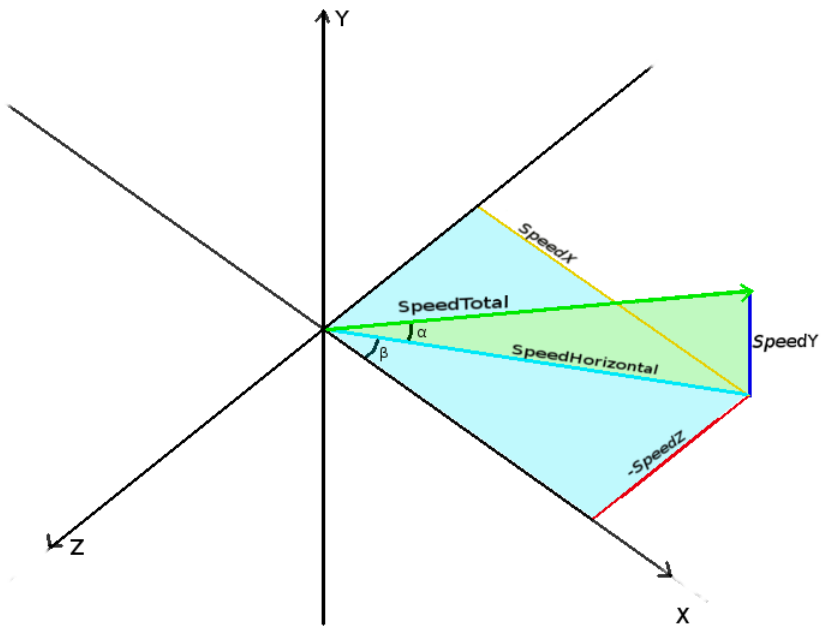


Figure 3.12: Speed calculations for a 3D model

3.5.2 3D Velocity Calulations

In order to calculate the velocity vector we need to know the velocity in x, z and y direction, as the airplane will move in all three directions in a the 3D environment. Having Total Velocity, Pitch and Yaw we need to use trigonometry to find the induvidual velocities. The following formulas found in Calculus 1[8] will be used:

$$Opposite = Hypotenuse * sin(\theta) \tag{3.1}$$

$$Adjacent = Hypotenuse * cos(\theta) \tag{3.2}$$

In order to calculate the speed in the y direction, speedY, we must set speedTotal as hypotenuse, use the α as angle, the opposite line will then be speedY. The triangle used to calculate this is marked with a green tint on Figure 3.12. Looking at the previous Figure 3.11, the α represent the pitch of the airplane, and will make them go up or down.

$$Opposite = Hypotenuse * sin(\theta) \tag{3.1}$$

$$speedY = speedTotal * sin(\alpha)$$

Before calculating the speed in the x direction, speedX, or the speed in the z direction, speedZ, we need to know the horizontal speed. This is marked as speedHorizontal in the figure. This is the adjacent line from the corner with the α angle:

$$Adjacent = Hypotenuse * cos(\theta) (3.2)$$

$$SpeedHorizontal = speedTotal * cos(\alpha)$$

Now to calculate speedX or speedZ we can see that the hypotenuse is SpeedHorizontal and the angle is β , in both cases. SpeedZ is the opposite line and SpeedX the adjacent. The triangle used to calculate this is marked with a blue tint, and is located between the light blue and the red line on Figure 3.12. Looking at the Figure 3.11, the β here represent the yaw, which will make them turn.

$$Adjacent = Hypotenuse * cos(\theta) (3.2)$$

$$speedX = SpeedHorizontal * cos(\beta)$$

The SpeedZ will be negative for beta between 0 and 180 degrees because the β goes in opposite direction of the z axis.

$$Opposite = Hypotenuse * sin(\theta) (3.1)$$

$$speedZ = -SpeedHorizontal * sin(\beta)$$

Having these three formulas, we could now calcualte the velocity vector.

3.5.3 Which SDK to use for Kinect

We are going to use either the official Kinect SDK or the OpenKinect[73]. The official SDK is maintained by the Microsoft Research team while the OpenKinect is an open source SDK maintained by the open source community. The main advantages of the two are[74]:

- The Official Kinect SDK is developed by Microsoft, who develops the Kinect as well. Meaning that they know the internal information about the device, while with the open source SDK we must reverse engineer it.
- Microsoft invests a lot of money in the Official SDK, which means that they will keep it up to date. At the same time we should not underestimate the power of the open source society.

- The Official SDK is well documented, and provides support forums, while OpenKinect does not have this feature.
- We have to calibrate the device if we are going to use OpenKinect.
- The OpenKinect supports more languages than Official SDK, such as Java and Python.

We decided to use the Official SDK as there is more support to be found to it from Microsoft, than from the OpenKinect community. At the same time, we are using Windows 7, with Visual Studio .NET, and these two have more integration with the Official SDK. The Official SDK does not need any calibration and also provides some examples and code. In addition to all this, the drivers for this are also automatically installed when plugging the Kinect onto the computer.

3.5.4 Mocking Kinect

One of our engineering tasks in the first sprint was to simulate and mimic the behaviour of the Kinect, more popularly called mocking. The reason for this was that we wanted to begin programming with the Kinect SDK, and testing its behaviour and features, already in the first sprint. As we did not receive the Kinect before sprint 2, mocking was an option.

Our goal was to determine if mocking Kinect was favourable, compared to the time we had disposable. We started to research which opportunities we had, and it quickly boiled down to two different options: Fakenect and Mockinect. Fakenect is a library which replicates the open Kinect SDK[76]. Since we had already decided to use the official Kinect SDK provided by Microsoft, Fakenect was not an option.

Mockinect on the other hand supported the official Kinect SDK3MocKinect. It was however still in the early phases of development, and did not support basic function such as recording skeleton information. The project had status as being in beta, it had few downloads, and the last update was a year old. At this point we decided that the benefits we would get from mocking Kinect would not outweigh the cost in time we would use to successfully set up the mocking framework.

3.5.5 Recognising Motions With Kinect

After we got the Kinect and its Framework up and running we found some developer tutorials from Microsoft and started our first Kinect Project. We managed to get the video stream on the screen and drawing a skeleton that tracked our movements.

In the first iteration of our Kinect prototype, we made a 3D box on the screen. We were going to make the position and the rotation of the box change. First its position was changed to where we tracked the users right hand in a 3D environment, and the box was also rotated depending on the position of the users left hand. The problem with this was that the tracking would happen all the time. So if you relaxed your hand or moved away from the Kinect the box would move accordingly.

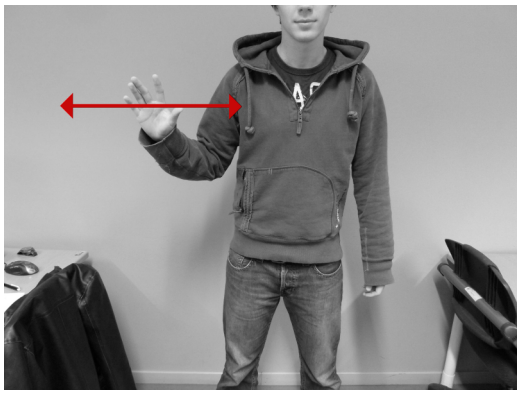


Figure 3.13: Gesture for moving sideways

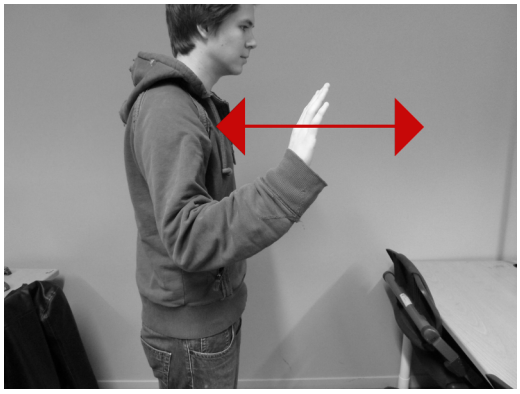


Figure 3.14: Gesture for moving forward and backwards

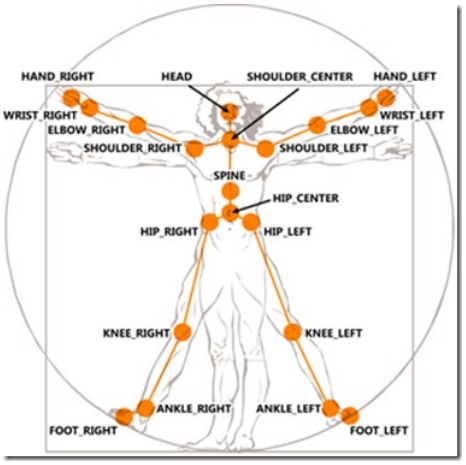


Figure 3.15: Different tracking locations for the Kinect. Figure made by Microsoft[53]

The hand movement where you start in one position, move the hand around, and finally stop is called a spatial gesture. We will just call it a gesture from this point on. We are going to try to capture some of the most basic gestures. The official Kinect SDK has no integrated solution for them. One of the basic gestures is panning. This is to move your hand in a straight direction, depicted in Figure 3.14. Another basic gesture is to just move your hand sideways, depicted in Figure 3.13.

Unfortunately the official Kinect SDK has only one tracker per hand as seen in Figure 3.15, and no integrated finger tracking. This will be a problem when we need to detect when a gesture is starting and when it is finished, since we can't track when a user is grabbing.

We needed another possible way to know when a user was starting or stopping a gesture. There were several possible solutions for this. Our criteria for choosing the best one was how hard it was to implement, and how good user experience it would give. The choice is important to discuss because it would heavily influence how we were going to make the Kinect work with our system.

Group 2 who had this course in 2011, had the same problem. The solution they used was to use a wireless mouse[7]. To start the gesture you had to click the mouse button and then release it when you wanted to end the gesture. This solution is very simple to implement, but we don't think it is very user friendly, because our user expects a speech and motion driven system without any other input devices.

Another possible solution we came up with was to give voice commands to start and end gestures. The good part of this solution is that it does not force you to use keyboard or mouse buttons. The bad part is that doing a lot of gestures fast would be exhausting for the voice. Also the time between each gesture would be longer, as clicking a button is faster than giving a voice command. When we have a working voice recognizer running

Table 3.5: Summary of implementation ideas and how well they fitted our criteria.

Name of idea	**Ease of implementation**	**User experience**
Clicking mouse to start and end gestures	Easy	Bad
Voice commands to start and end gestures	Hard at the moment	Partially good
Predication start and end gestures	Hard	Good
Track hands over a height limit	Easy	Partially good

this solution would be simple to implement. But we were planning to add gestures before voice control. We might look further into this solution after we have implemented the voice control part of the system.

The most professional solution is to do heavy calculations in each frame to predict if it is a gesture start or gesture end. This would mean that the user does not need to signalize that the gesture is starting. This is used in alternatives to the official SDK like the iisu SDK[52]. One way to make this predictions is by using Hidden Markov Models as discussed in a project at Standford[51]. This solution is very costly for the performance, because you have to compare every frame to some subset of frames before. It is also very complex and would be very time consuming to create for the first time. For this project we can not devote that much time into making prediction algorithms.

We got an idea for a new solution; have your left hand in the air when doing a gesture with your right hand. This solution did not work well for two reasons; It was exhausting to have your left hand in the air for a longer period of time, and by using the left hand to start and stop we only had one hand left for both rotation and movement. So we could not use this solution.

However we got a new idea using the core of this idea. The idea was to only track the hands when they were higher than a defined (and easy changeable) limit/line. This solution worked very well. It was not too hard to implement and you could both rotate and alter the position of the box. We decided to make a simple camera model and replace it with the box to better show that the object was rotating and moving in the right direction.

For this decision both criteria were very important. We could not make a system with great user experience if we did not have time to make it, but we needed a system which at least gave a partially good user experience. From Table 3.5 we can see that idea number 4 is the only one that satisfies both the criteria about providing a good user experience and not being to complicated to implement. Based on this knowledge, we decided to implement that idea.

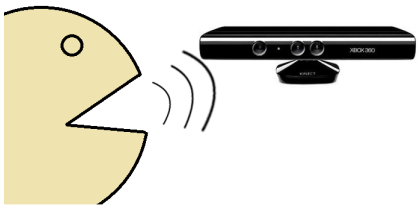


Figure 3.16: Recognising voice with Kinect

3.5.6 Recognising Voice With Kinect

On of the parts of our project revolves around speech recognition. We decided to dedicate some time to research on this field, in order to know what we could expect in later sprints.

In the beginning we wanted to learn about other peoples experience with the Microsoft.Speech framework, so we started to read through guides and tutorials we found on the Internet. There were a lot of positive feedback about this framework[78], but unfortunately most of the guides we found were not compatible with our XNA project. Some of them were written for other project types, like Windows Forms, which has a different data flow. Other guides were simply outdated and made use of now deprecated functions. We then turned our attention to Microsoft's own documentation pages, which had a couple of guides on how to create basic grammars and recognise words from audio files. Combining this with the guides we already had found, we managed to make our program recognise easy words like "up", "down", and "plane".

The voice recognition part of the system would be easy for the user to use. The user should only stand in front of the Kinect device and speak, like showed in Figure 3.16

Chapter 4

Requirement Specification

In this chapter about requirement specification we describe and discuss the requirements the customer and the team developed for the system. The chapter is split into different sections that all introduces the requirements for the system in a different way. The chapter starts with a section describing the different user groups for the system in a thoroughly way. The rest of the chapter contains different sections describing the requirements through the functional and non-functional requirements and the use cases, both textual and diagrams. At the end, we present the some of the user stories from the product backlog.

4.1 Description of the User Groups

Since our system is only a prototype, the customer is the only user group. They will look through the code and maybe develop it further. If the system should be made into a finished air traffic controller product, there will be a different user: the air traffic controller.

The Evaluator / Customer

The person who evaluates the prototype needs to have software development background, and will therefore look at all the aspects of our product, not only the user interface. He requires clean and easy-understandable code, which will be easy to modify and develop further. The graphical user interface has a lower importance than the functionality of the system, especially the user interaction with the system through motion and voice are important. Since the evaluator is the one who presented us with the task, he will have a clear expectation of how he wants the system. Therefore, it is important that the system fulfills all of the requirements that has been specified.

The Air Traffic Controller

The air traffic controller is a person who works in the control tower of an airport. He has a lot of knowledge about how the airspace should best be controlled, but he is not necessarily an expert computer user. He will often work in a stressful environment, where decisions must be made at all time to prevent potentially dangerous situations.

This person would want a system which is easy to use, which presents a good, correct view of the airspace, the runways and which is secure and reliable, and of course that works without any errors.

4.2 Functional and Non-Functional Requirements

In this section we will list the functional and non-Functional requirements for the system. The textual use cases are supported by the functional requirements stated in this section. One functional requirement can be the support for several textual use cases. The non-functional requirements are requirements for the systems usability and scalability, among others.

Functional requirements describes what the system must do. The requirements do not go in any detail of how the system should be implemented or how it should be used, they are only requirements for the functionality. The system's main functional requirements consists of displaying the airspace as a 3D model, being able to control the model using motion and being able to control the airplanes in the model using voice. The functional requirements are shown in Table 4.1.

Non-functional requirements describes how the system should be, rather than what it should do. They describe requirements that are hard to measure for example usability, but they also consist of requirements regarding the more practical aspects like documentation. The system's most important non-functional requirements are that it should be well documented, it should be easy to test, it should be easy to expand with new features, and the code in the system should be easy to understand. The non-functional requirements are shown in Table 4.2.

4.3 Use Cases

This section contains the use cases we have made for our project. The textual use cases are supported by the functional requirements stated in section 4.2, one functional requirement can be the support for several textual use cases. Further in this section, we have made textual descriptions of each of the actors, in addition to the description of all the use cases.

Table 4.1: Functional requirements

ID	Description	Priority
FR1	The system should present the airspace as a 3D model.	High
FR2	The user should be able to change the viewpoint of the model using hand gestures.	High
FR3	The user should be able to control the airplanes in the model using speech.	High
FR4	The system should display all the airplanes in the airspace.	High
FR5	The system should display the names and the coordinates of the airplanes.	Medium
FR6	The system should display information about selected airplane and possible commands.	Medium
FR7	The system should notify the user of airplanes on collision course.	Low
FR8	The system should provide the user with audio feedback.	Low

Table 4.2: Non-functional requirements

ID	Description	Priority
NR1	The system should be well documented.	High
NR2	The system should be easy to test.	High
NR3	The system should be easy to expand with new features.	High
NR4	The system should be written in easily understandable code.	High
NR5	The system should respond to user input in 100 ms.	Medium
NR6	The user should be able to use the system for ten minutes without making any errors after reading the manual and receiving training in the system for half an hour.	Medium
NR7	The system not experience loss of performance with FPS below 50 when running continuously for three hours.	Medium
NR8	The system should be able to run for 30 minutes without any errors.	Low

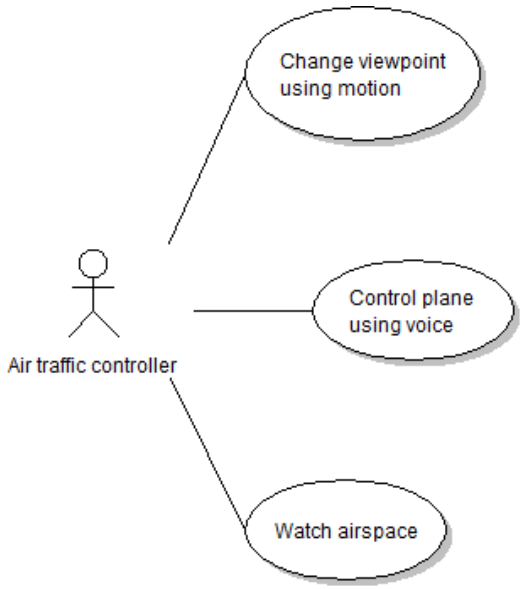


Figure 4.1: Use case for the air traffic controller

4.3.1 Use Case Diagrams

The use case diagrams are shown in Figure 4.1 and in Figure 4.2. Here, the actors are represented as stick-figures, and the use cases as ovals.

The use cases represent the stakeholder/actor goals. A use case defines the interaction between external actors and the system under consideration to accomplish a goal. The actors need to be able to make decisions, but they do not need to be human. An actor can be a person, company, computer program, system, HW, SW etc.

4.3.2 Textual Specification of Actors

Each actor description contains the actor's name, its description, examples of different actions and a classification of how complex each actor is. The classification is calculated by the rules in the compendium [1]:

- **Simple actor:** Relates to a single system with a defined programming interface.
- **Average actor:** Communicates with the system or another actor using communication protocols or a textual interface.
- **Complex actor:** Communicates via a graphical interface.

The textual specifications of the two actor are shown in Table 4.3 and in Table 4.4.

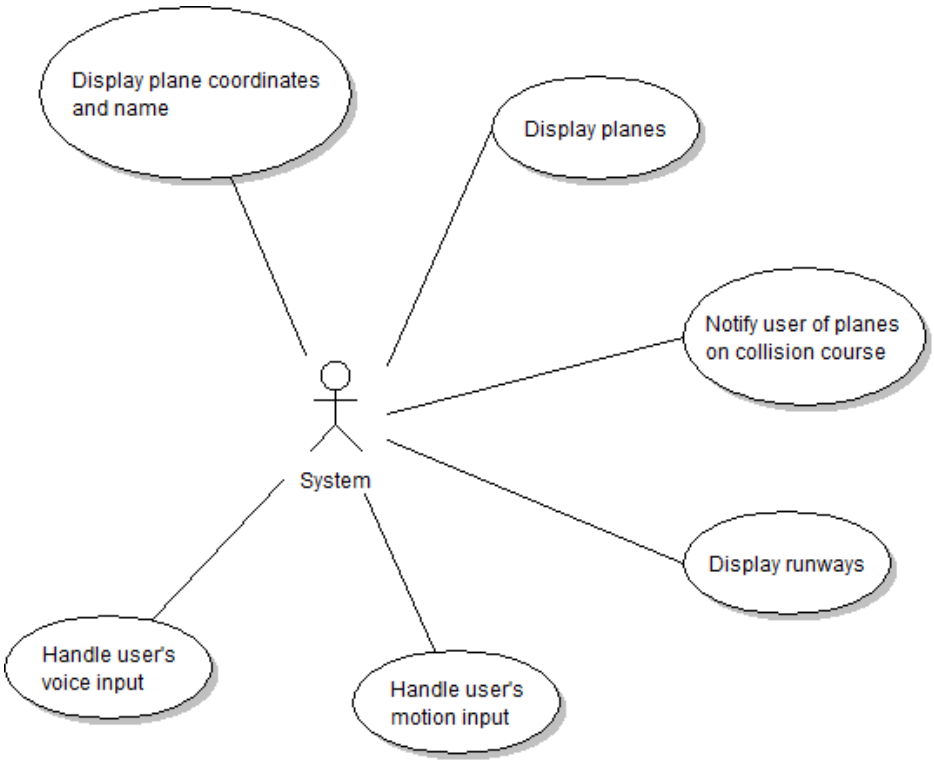


Figure 4.2: Use case for the system

Table 4.3: Textual specification for the air traffic controller

Actor ID	Air traffic controller.
Description	Is the main user of the system and the one who has control over the airspace.
Examples of actions	• Can look at the 3D model to get a complete overview of the airspace. • Can control the 3D model using motion gestures. • Can control the planes in the model using gestures.
Classification	Complex

Table 4.4: Textual specification for the system

Actor ID	System.
Description	Shows the 3D model and handles the input from the user.
Examples of actions	• Displays the model with the moving planes. • Notifies the user if two planes are on collision course.
Classification	Complex

4.3.3 Textual Use-Cases

The textual use cases consists of the use case's name, the related actor, the trigger event for the use case, its post- and preconditions, its event flow (both normal and modified) and its classification. The classifications are as follows:

- **Simple use case:** 3 or fewer events.
- **Average use case:** 4 to 7 events.
- **Complex use case:** 8 or more events.

The textual use cases are shown in item 4.5 to item 4.11.

4.4 The Product Backlog

This section contains some general information about our product backlog. In addition, it describes the four user stories from our product backlog, that were critical to get done if we wanted to solve the given task.

4.4.1 Description of the Product Backlog

The complete product backlog can be found in Appendix F. The produckt backlog contains 17 user stories. The user stories have an overall priority, either high, medium or low. Nine of the user stories are ranked high, three are medium and five are low. The high ranked user stories were important that we finished, and those we should do first. The medium ranked user stories should also be done to create a good program. The low ranked user stories should only be done if we were finished with all the other user stories.

We managed to do all the user stories in the backlog. During our work with the product, some user stories were moved and other were added. Also, the prioritization of some of them were changed. This is the reason for the user stories not being numbered from 1 to 17, and that they are not in ascending order.

Table 4.5: Watch air space.

Use case name	Watch airspace.
Actors related	Air traffic controller.
Trigger	The air traffic controller starts the program.
Pre-conditions	The program is shut down.
Post-conditions	The model are displayed with all the airplanes visible.
Normal event flow	1. The air traffic controller starts the program. 2. The model shows. 3. The air traffic controller looks at the model.
Variations in the event flow	
Classification	Easy

4.4.2 Critical User Stories

The four critical user stories are shown in Table 4.12. The first one is US2, and it is about the 3D model. This model was important to create first since all our further work depends on a working 3D environment.

The next important user story, US4, is about the documentation. Since the documentation was really important in this course, this user story got a high prioritation, and we spent a lot of time on it.

The US7 is about integrating motion and Kinect with the already created 3D model. This was another of the main requirements set by the customer, and it was vital to fulfill.

The last shown user story is US8, one of the two user stories regarding voice input in the product backlog. Voice input should be adde to control the airplanes in the model, and is the customer's third important criterium from the task.

Table 4.6: Display the airplane's coordinates and names

Use case name	Display airplane coordinates and names
Actors related	System.
Trigger	The user starts the program.
Pre-conditions	The program is not running.
Post-conditions	The planes with names and coordinates are displayed in the model.
Normal event flow	1. The program is started. 2. The model is initialized. 3. All the planes are shown with their name and coordinates displayed above them.
Variations in the event flow	2a. The model is not properly initialized. 2a1. The system terminates, and then starts again from point 1.
Classification	Average

Table 4.7: Display the airplanes

Use case name	Display airplanes
Actors related	System.
Trigger	The user starts the program.
Pre-conditions	The program is not running.
Post-conditions	The plane is displayed at its current placement in the model.
Normal event flow	1. The user starts the program. 2. The system initializes the model. 3. The airplanes are displayed.
Variations in the event flow	2a. The model is not properly initialized. 2a1. The system terminates, and then starts again from point 1.
Classification	Average

Table 4.8: Notification if airplanes are on collision course

Use case name	Notify user if airplanes are on collision course.
Actors related	System.
Trigger	Two planes are moving towards each other, and are going to crash.
Pre-conditions	The program is running.
Post-conditions	The system is indicating which planes that are at risk by flashing and/or sound.
Normal event flow	1. The system checks all the planes in the air and calculates their speed, altitude and direction of movement. 2. The system discovers that two planes are on a colliding course. 3. The system provides audio feedback, saying which airplanes are on collision course and the time before a collision will occure.
Variations in the event flow	
Classification	Easy

Table 4.9: System audio feedback

Use case name	System should provide audio feedback
Actors related	System.
Trigger	The user starts the program.
Pre-conditions	The program is running.
Post-conditions	The system provides audio feedback
Normal event flow	1. The user starts the program. 2. The user provides the system with a voice command 3. The system responds with audio feedback.
Classification	Average

Table 4.10: Handle user's voice input

Use case name	Handle user's voice input.
Actors related	System.
Trigger	A user speaks into the microphone.
Pre-conditions	The program is running.
Post-conditions	The action corresponding to the users commands is performed.
Normal event flow	1. The system receives the sounds from the microphone. 2. The system checks the sound against its internal stored commands dictionary. 3. The system executes the action corresponding to spoken words.
Variations in the event flow	3a. The model is not properly initialized. 3a1. No change happens. 3a2. The use-case starts over from point 1.
Classification	Average

Table 4.11: Handle user's motion input

Use case name	Handle user's motion input.
Actors related	System.
Trigger	The user moves in front of the camera.
Pre-conditions	The program is running.
Post-conditions	The action corresponding to the users motion are executed.
Normal event flow	1. The user moves in front of the camera. 2. The system register the motions. 3. The system tries to find an event corresponding to the movement. 4. The event is executed.
Variations in the event flow	3a. No corresponding event is found. 3a1. No change happens. 3a2. The use-case is restarted from point 1.
Classification	Average

Table 4.12: The critical user stories from the product backlog

ID	**User story**	**Overall priority**
US2	As an air traffic controller, I want to see moving airplanes on my screen in a 3D model, so that I know where the plane are in the air	High
US4	As a student, I need to document the work thoroughly, so that we can write a good report from our project	High
US7	As an air traffic controller, I want to use hand gestures to change the camera direction and position, so that I can control the system without a keyboard	High
US8	As an air traffic controller, I want to give speech commands to select an airplane, so that I can give it more commands later	High

Chapter 5

Overall System Design

This chapter provides an overview of the architecture of the system. The architecture is presented using multiple descriptions, views and figures. The first part of the chapter gives a description of the architectural drivers for the system and related tactics. We then present the patterns that have been chosen for the system and reason for it. We also provide different views of the architecture and system. For this, we use the so called 4+1 architectural view model, to be able to describe the system from the viewpoint of different stakeholders

5.1 Architectural Drivers

In this section we will discuss the architectural drivers that have had the biggest focus during this project. Together with the customer we agreed and prioritized four quality requirements. These should be thought about while planning and writing code, to make sure that the final system are as good as possible.

The four quality requirements were performance, testability, usability and modifiability.

Performance was very important for the customer. The system must be able to show a 3D environment with multiple planes on the screen. At the same time it must be able to handle motion detection and voice recognition.

Testability was also of importance. The system should work without any big problems, as any error could be catastrophic in a real system. In our case where we only have implemented a prototype, it was not as crucial, but still important. In order to have few errors we needed good testability.

Usability is important for our system, and we had that as out third driver. The customer wanted a system that was intuitive and easy to use. Most people are not familiar with using a system without keyboard, mouse and buttons. So we needed to make the transition as easy as possible for them.

The last driver was modifiability. Since we should extend our system in each sprint, it

was important to the team that the system was easy to modify along the way. After this course we will not work on the system anymore, so it was also important for the customer that the system offered the possibility for another team to continue working with our code base.

Other driver that was important in our project and for the group were the desire to get a good grade, experience and deadlines.

During the project our group stayed focus to maintain all deadlines, we focused on making a good product that pleased the customer, documenting our work in a good way and presenting it in such a satisfactory manner that we would result in a good grade.

The experience in the group with similar work was different among the different group members. Some had had more experience with such teamworks, this kind of programming and the frequent documentation, while others did not. It was therefore important to identify weaknesses, strengths and expectations early on.

The project contains several deadlines and a time limitations that all needed to be kept. This required us to make good and realistic plans for our work and time disposal.

5.2 Architectural Tactics

In this section we will discuss the architectural tactics we can use to make sure we get high quality on the most important attributes. We used tactics from a presentation by Alf Inge Wang in TDT4240 Software Architecture [13]

5.2.1 Performance Tactics

Our goal by using tactics to achieve good performance was to make sure that the system would respond quickly to user input, and not have any troubles with drawing in 3D.

Resource demand is a performance tactic we are going to use. We are going to reduce the number of events processed. We found that the velocity of the airplanes in all x, y and z directions will be changed very infrequent. Our prototype system does only use acceleration when a command for speed change is called. At first we calculated the velocity in each direction every frame, later we changed the calculation to only happen if the total speed or direction of the plane changed. This is similar to the model rotation matrix that will only change when the direction of the model is changed. This is very infrequent as well, so we improved it by only calculating the Matrix if the direction changes.

5.2.2 Testability Tactics

By using tactics to achieve testability we made sure that the system worked correctly and that no errors would happend. It should be easy for us to test the system during the project, and for the customer to test the system afterward because they are not in Trondheim during the implementation phase of the project.

Internal monitoring is a testability tactic we will use. We are testing that the performance is not reduced by tracking frames per second. If the performance is good, this number will be towards 60, which is the upper default limit for XNA.

We also used the tactic *single responsibility* [46], by dividing classes that had too much responsibility. This means that the classes where more easy to test because they were smaller. We reduced the work done in the constructor methods. We tried to avoid making new instances of other objects in the constructor, so you did not need to make many instances to test one class. This lead to faster and less complex test.

5.2.3 Usability Tactics

Our goal by using usability tactics was to make sure that the system was easy to use for the user. Having a system without keyboard and mouse will most likely be a new experience for most users, and we needed to make sure that they can handle it without too much trouble.

To maintain a model of the task, the user and the system, is a usability tactic we will use. This is done to make sure that we do not forget the needs and the technical understanding of the user.

We also used *design-time tactics*, like separating the Kinect user input from the rest of the application. It would then be easy to refine how the control is handled after input from users. This would lead to a good user experience.

5.2.4 Modifiability Tactics

To achieve modifiability we needed tactics to make sure that the system was easy for us to change if needed. Because we are working iteratively, it is very likely that we need to do some re-factoring of the code along the way, so this should not be too hard. It should be easy for a new team to take over the code base, because we are not going to work more on the implementation after this course.

Prevent ripple effects is one of the modifiability tactics we used. We tried to reduce the number of other classes affected by a change in one class. We did that by having controller classes between the model classes and the input classes, which would contribute to changes in them. For instance, by having an KinectInputVoice class, an AirplaneController and an AirplaneModel. A change in KinectInputVoice would not make any changes to the AirplaneModel and the other way around.

We also used the tactic *easy component replacement*, to make sure that components is easy replaceable. The previous example also shows this. We can replace the whole KinectInputVoice without making changes to AirplaneModel.

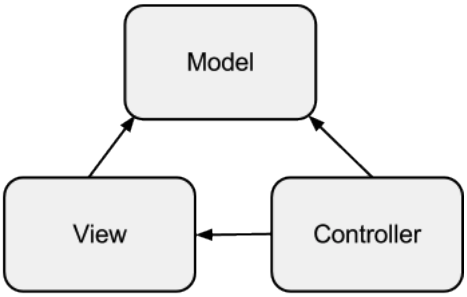


Figure 5.1: Architecture using MVC

5.3 Architectural Patterns

This section contains a description of the three different architectural patterns we considered for our system. The three different patterns were model-view controller, layered architecture and client-server, and we read about them in Software Architecture in Practice, Second Edition [2].

5.3.1 Model-View-Controller

Model-view-controller (MVC) is an architectural pattern with focus on separation of concerns [2]. It is highly popular in many different kinds of systems. In MVC the components are divided in 3 main packages: The model, the view and the controller. The model package have classes that holds data, the view classes displays some or all the models classes, and the controller classes manipulates the classes from the other two packages. It should make it simpler to test each component on its own, and make the system easier to implement and maintain in the future. In figure in Figure 5.1 you can se a figure of the MVC architecture.

5.3.2 Layered Architecture

Layered architecture is another architectural pattern with focus on separation of concerns [2]. Here the system is divided into different levels of abstraction. This pattern is good for large systems that should work in different hardware or environments. Each level can be made by a different team, be modified or changed completely without changing too much in other layers. An example of layered architecture is shown in Figure 5.2

5.3.3 Client-Server

The client server architecture are an architecture consisting of a centralized server that contains shared data, programs, a database or something else that is common for the systems working towards the server [2]. A client is a computer that connects to the server

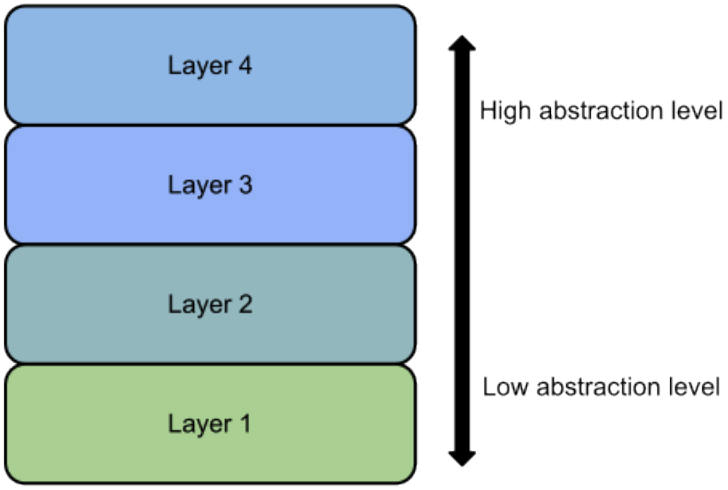


Figure 5.2: Architecture using the layered pattern

and uses data, programs etc. from it. Usually, many clients are connected to the same server at the same time. The clients communicate with the server via a network. This architecture makes it possible for several clients to share the same data, and also for change in one client to be distributed to all the other clients, via the server. See Figure 5.3 get an idea of the client-server architecture.

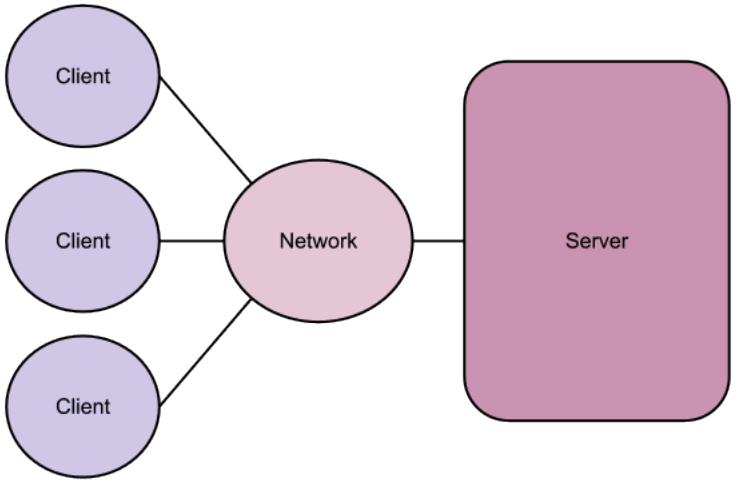


Figure 5.3: Architecture using client-server

5.4 Architectural Rationale

Since we are only going to make a small prototype of the system, the requirement to the architecture are somehow different than if we were to implement a system that should actually be used for controlling a real airspace. The differences of the two systems are concerns about security and concerns about numbers of (simultaneous) users. Also, the amount of data in such a model will be much higher if the system should be used in a "real environment".

Choice of Architecture

We choose to use the MVC architectural pattern. We wanted to learn more about it and the features should be useful for the project. In this case it fits with our Scrum planning as well, as we can make some parts of the system in the beginning and make components for movementDetection and speechRecognition later.

The MVC pattern can be adapted to our program by putting all the classes that handles the graphical user interface in the view part, the "objects" or models in the model part, and all the classes that manipulates the other classes in the controller part.

Some of the nice features of layered architecture is implemented in the framework for C# already, like some hardware and environment abstractions. Still, we feel that the MVC would suit our system better, both with the agile development method we will be using, and the possibility to extend our initial model with control options.

In a real-word system a client-server would fit this project well, however for the prototype we made it was not necessary, as it was a single program, running on a single machine without communications with others.

5.5 Design Patterns

Design patterns is a reusable solution to a commonly occurring problem [14]. They describe spesific objects and classes, and is thus less general than Architectural Patterns. We used singleton pattern, which is described in this section.

5.5.1 Singleton

Singleton is a creational type of design pattern[14], and is used to limit the number of instances of a class. We used it to make sure that only one instance of the display is made. Thus we knew that all classes that use the display will modify the same instance. Using singleton we managed to prevent ripple effects, since we did not need all the classes that would use the display, to take the display instance as a parameter in the constructor. Preventing ripple effects is one of the modifiability tactics described in 5.2 Architectural Tactics.

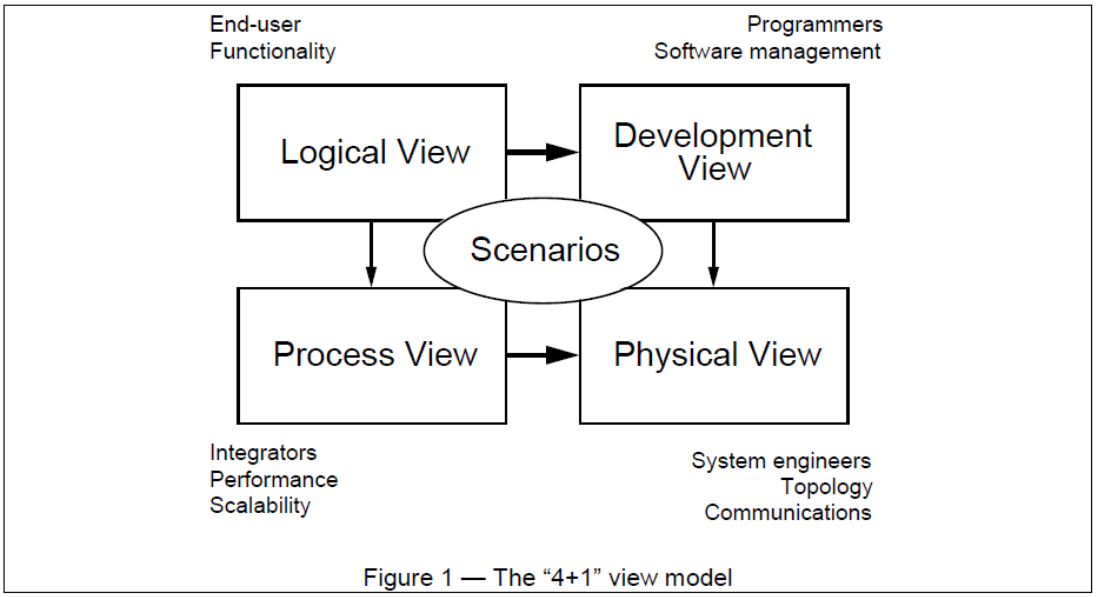


Figure 5.4: The 4 + 1 view model [6]

5.6 Views

The 4+1 view model is an IEEE standard [6] which describes an architectural model composed of four different views and some scenarios, as shown in Figure 5.4. The views that are described in the model are the logical view, the process view, the physical view and the development view. We have created all these four views for our system. One way to represent scenarios are Use Cases. Our textual Use Cases can be found in subsection 4.3.3.

5.6.1 The Logical View

A logical view's purpose is to support the functional requirements. It is an object-oriented decomposition of the system, and shows principles like abstraction, encapsulations and inheritance. An example of a logical view is a class diagram. Here, related classes can be grouped together, and classes that communicates with each other is connected with an arrow.

Our logical view is a static structure diagram, showing all the classes in our system divided into packages. The view is shown in Figure 5.5. Since we have chosen the model view controller (MVC) architecture pattern, all of our classes are a part of either the model, the view or the controller, except for the main program, which is the class that initializes

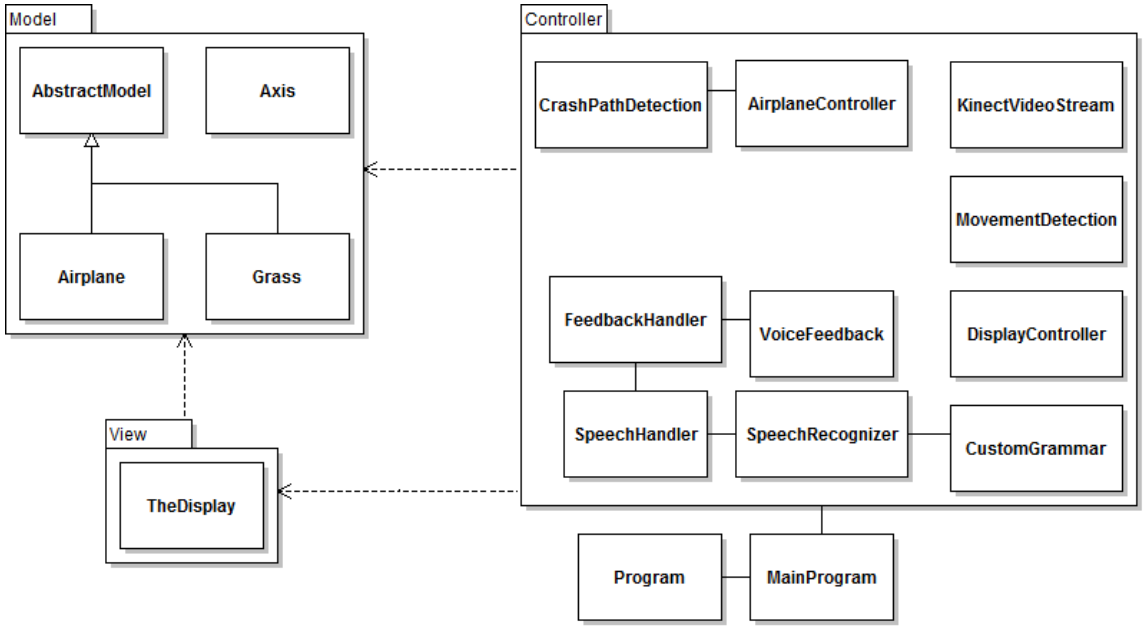


Figure 5.5: The logical view: The static structure diagram

the whole system.

The model part of our system has three classes, the Airplane, the Axis, and the Grass, in addition to one abstract class, the AbstractModel. The most of these classes contains a 3D model or some graphic. They also have methods to modify the object in some way.

The view part of our system consists of only one class, the Display. This class is responsible for the graphical user interface. It draws everything to the screen, making it visible to the user.

The last part of our system, the controller classes, are the classes that handles user input and controlles the other classes. When the user has an input, like voice or motion, the controller classes handles it. This could be by changing variables in the model classes or changing the view point of the view part of the system.

We have one abstract class, the AbstractModel. The reasons for choosing to use abstract classes were that this ensures that all the classes that inherits from this class, has to implement the abstract methods defined in the class. Also, if we are going to make changes to our models, we can make changes in the abstract class, forcing the changes to happen in all of the classes that inherits from this class. Another nice feature with an abstract class is that it is easy to add new children with the same methods.

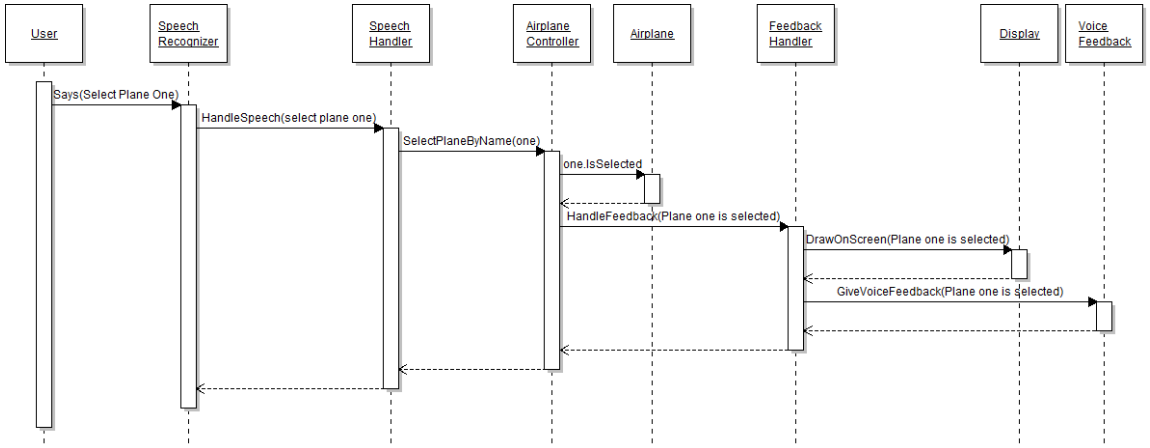


Figure 5.6: The process view: The sequence diagram showing the selection of plane one

5.6.2 The Process View

A process view is used to group tasks together that forms a process or an executable unit. A single process is defined as concurrent threads running and exchanging information. Each process is based on some task that the program should handle, showing how the messages goes between many threads, and starts and terminates them as they receive a new message or sends a return statement. An example of a process view is a sequence diagram. Here, the user makes a request, and the program executes it, showing the message flow as this is done.

For the process view, we have created a sequence diagram. The sequence we have looked at is a user talking to the system to be able to select a plane, here, "plane1." From Figure 5.6 we can see that the message from the user is interpreted by the controller classes SpeechRecognizer and SpeechHandler, then passed to the model class Airplane via AirplaneController, before the output finally is presented to the user via the view, the Display and as voice feedback, the VoiceFeedback. The FeedbackHandler is the class responsible for passing the feedback on to the two different feedback classes.

5.6.3 The Physical View

The purpose of the physical view is to present the system engineers with a model showing the physical mapping of the software onto the hardware. This is done to make them aware if which hardware and software that are needed, and how they are related to each other.

Our physical view are shown in Figure 5.7. The system's hardwares are shown at the bottom. This is a PC and a Kinect. Above the hardware, our operating system is running, which is Windows 7. Then, the .NET framework and the XNA framework follows. At the

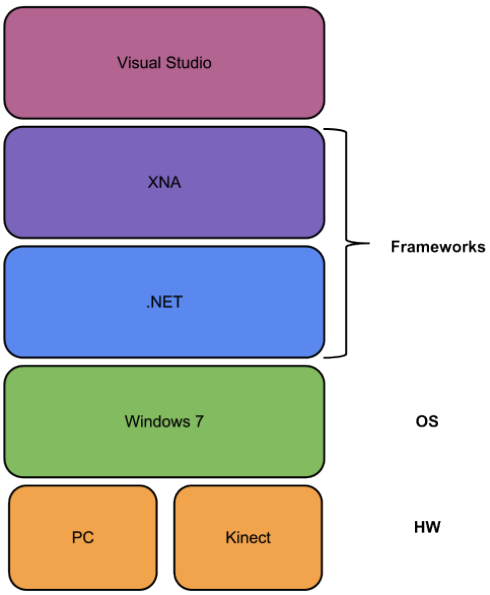


Figure 5.7: The physical view. The figure is based on a lecture in video games[12].

very top of our figure, we have placed Visual Studio, our main development environment.

5.6.4 The Development View

The development view is created to help the programmers and the software managers to decide which part of the system should be implemented first, second and so on. The view should describe wich parts of the system that depends on each other, presented in the view as several layers. It should also show which part of the system that can be developed as stand alone packages.

As a development view, we have chosen to draw a figure describing the order of the implementation of the different parts of our system. This is shown in Figure 5.8. We should start with making the 3D model of the airspace, then add the possibility to control the model using the keyboard's buttons. Next, we should add the camera, the motion control of the camera, and then end our implementation with the ability to control the planes in the model using voice. A thing that is important in our system, is that each of the parts is an extension of the previous system, making the implementation order important.

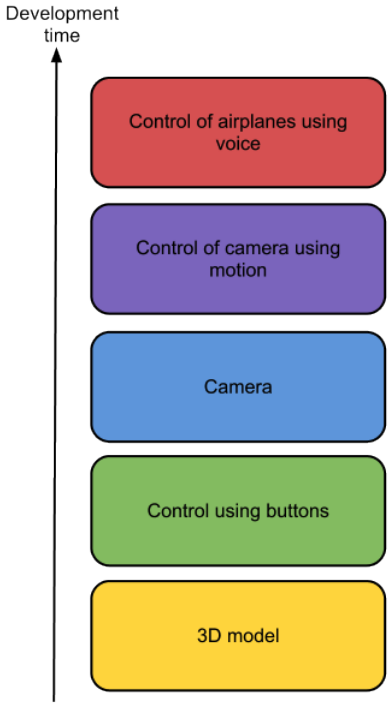


Figure 5.8: The development view

Chapter 6

Overall Test Plan

The overall test plan consists of all the tests that should be executed on the system before we deliver it to the customer. Some of the tests have a priority describing how important it is that the test are passed. The test cases are developed from the systems requirements. An overview of the different testing methods and types can be found in section 3.3

The overall testing plan is based on IEEE 829 test plan structure[22], but has some inequalities, mainly because the IEEE 829 is a structure suggestion for a more comprehensive test plan than the one we will make in this project. Appendix H presents the level test plan outline, taken from IEEE 829, showing the elements suggested in a more comprehensive test plan. In our test plan we will focus on describing the following:

- Test items and their identifiers
- Features tested
- Testing approaches
- Pass/fail criteria
- Environmental needs
- Test responsible
- Test Schedule

The conducting of the tests and the results can be found in chapter 12

6.1 Test Items and Identifiers

All our tests were based on the requirements that we made in the beginning of the project. While the unit and integration tests were made during programming.

For the test case IDs we used a capital letter from the first letter of the test type followed by a number. Example for a system test: S01, usability test: U01, etc.

6.2 Features Tested

The goal for our group was to test as much as possible of the system, making it ready for the final delivery. This meant that we should have tested the system from the smallest component to the system as a whole.

We chose to test the system from methods in the code to input sent from the user to the system, thus unit (white-box)- to system (black-box) tests.

6.3 Testing Approaches

In our system we used white-box testing for the unit-tests, grey-box testing for the integration tests and black-box for the rest.

Organization of Tests

To perform the white- and grey-box tests the tester should have had read and understood the system, the requirements and the test cases. The testers should also read through the manual to get to know how the different gestures has to be performed. For the black-box tests, like the system test and the usability tests, the tester has to get an introduction to the system, primarily through a manual. Also some of the gestures should be explained by a person already familiar to the system.

The input for the different tests were different gestures and voice commands performed in front of the Kinect. The output was to see the result on the screen. For an overview of the preparation, execution and results of the different tests see chapter 12

Reasons for our Choice

We chose to perform both grey-and black-box testing of the system to get as much coverage as possible, to be able to start testing early and offer the ability for all types of users and peoples to test the system. A more thoroughly reasoning for our testing choices can be read about in section 3.3.

6.4 Pass/Fail Criteria

For an item to pass it has to fulfill the output specified in the test case, an item failed if the output was very different from the expected output. For the unit and integration tests the output had to be right 100% of the time for the tests to pass.

6.5 Environmental Needs

To perform the different tests we need to have some minimum requirements for the system [68], both the the hardware the system will run on, and to the software on the system. We do also need some requirements regarding the persons who are to conduct the tests.

Hardware Needs

- Microsoft Kinect sensor
- 32-bit or 64-bit processor
- Dual-core 2.66GHz or faster processor
- Dedicated USB 2.0 bus
- 2GB RAM
- Graphics card that supports DirectX 9.0c and Shader Model 1.1

Software Needs

- Windows 7
- Microsoft Visual Studio 2010
- .NET Framework 4.0
- Microsoft Kinect Software Development Kit
- Microsoft Speech Platform SDK v11
- XNA Framework

Personnel Needs

All tests can be performed and executed by all team members. The acceptance test should be performed by the customer, or by a team member with the customer present. The usability test should be performed by a user who has not been developing the system.

Other Needs

To avoid problems when using the Kinect there are some needs that need to be fulfilled in the room one is going to use the Kinect in.

- For the camera to detect that a person is standing in front of it, there must only be one person in front of it.
- There has to be sufficient light in the room, so the camera sensor can detect/see movement
- The sensor detects best and most accurate when there is no background noise behind the person.

6.6 The Test Responsible

The test manager was the head responsible for assigning people to execute the different tests described in the test cases. All the programmers were responsible for their own code writing and to make sure their code worked the desired way.

6.7 Test Schedule

Shows when the different tests were executed

Table 6.1: Test schedule

Test ID	**Test name**	**Date executed (dd/mm)**
S01	Change camera viewpoint	12/11
S02	Change camera zoom	12/11
S03	Change camera rotation	12/11
S04	Select desired airplane	12/11
S05	Change airplane heading	12/11
S06	Change airplane pitch	12/11
S07	Show information	12/11
S08	Follow airplanes	12/11
S09	Airplanes on collision course	12/11
P01	System responding	12/11
P02	System uptime	12/11
P03	FPS	12/11
P04	15 airplanes in airspace	12/11
U1	Usability	31/10

Chapter 7

Sprint 1

This was our first sprint, and after 2 weeks of pre-study we were eager to start implementing on a shared code base. The group decided to have Joachim as Scrum Master for this sprint, because he was one of the people with experience from being Scrum Master before this project started, and the rest of the team wanted to see how it was done before trying themselves.

This chapter is divided into the following parts. The sprint breakdown with all the meetings and important dates. The sprint planning with our overall plan for this sprint, and the user stories we are going to do. The chapter about testing, where we discuss how we tested the tasks and user stories we finished. The results chapter where write about what we finished, with some screenshots of our system. Finally, the sprint retrospective with the hours we have used on each task, the burndown chart and reflection.

7.1 Sprint Breakdown

The sprint started on Thursday 6th of September with a planning meeting at 08:00 with the team, that extended into a planning meeting with the customer at 09:30. Later the same day we had an advisor meeting at 13:00. Every day we were going to have a daily scrum meeting for about 10 minutes with the team members.

We were going to work together on Tuesday, Thursday, half of Friday and half of Wednesday. At the middle of the sprint, at the 13th of September, we were having an advisory meeting. At the end of the sprint, the 20th of September, we were having a sprint demo and review meeting with the customer at 09:30. After this meeting we ended the sprint with a sprint retrospective meeting at 10:30.

Table 7.1: The current user stories in the sprint backlog

ID	User story	Overall priority	Priority this sprint
US1	As a programmer, I want a good and fast testing environment, so that I know that the code is correct after implementing and refactoring.	High	High
US2	As an air traffic controller, I want to see moving airplanes on my screen in a 3D model, so that I know where the planes are in the air.	High	High
US3	As a programmer, I need to learn the Kinect framework, so that I can implement the system to support Kinect.	High	Low
US4	As a student, I need to document the work thoroughly, so that we can write a good report of our project	High	High

7.2 Sprint Planning

Our plan was to make the fundamental part of our system work. This is the part of the program that contains airplanes moving in a 3D environment. It was important to set the testing environment up and running as soon as possible and improve our testing knowledge, so we could easily work with unit and functional testing.

We were also going to do a lot of documentation in this sprint, and were planning to finish up an initial draft of the first four chapters of the documentation. This includes writing it in LaTeX, which we assumed was going to take a lot of time. If there was time left we were going to research the Kinect framework, and try to better understand the amount of time this phase would take, so we could improve our estimates for later.

First we divided the user stories from Table 7.1 into smaller engineering tasks (ET) and then we used planning poker to assign them story points. These estimations were a bit hard to make because of variety of knowledge about code and testing, and somewhat because we all did not have time to work with everything during the pre study phase. We usually listened to the more experienced person in the given field when we had differences in the estimations.

We did also make some acceptance criteria. It was our understanding from the customer that he wanted us to do these plannings before the meeting with him. When we met, we went through the plans, and got confirmations from the customer that everything looked good. They agreed with the plans, but meant we should divide the tasks even more, and

Table 7.2: The acceptance criteria and their evaluation

ID	**Acceptance criteria**	**User story**	**Verified Finished**
AC1	All programmers should be able to make tests	US1	Verified by group
AC2	Having a testing environment for unit and functional testing	US1	Verified by group
AC3	All planes moves on the screen, in the direction they are pointing	US2	Verified by customer
AC5	Being able to change to direction and velocity of planes	US2	Verified by customer
AC6	Create a simple project which captures basic Kinect gestures	US3	Not done
AC7	Decide on which framework to use for Kinect	US3	Verified by group
AC8	Decide whether to mock Kinect is favourable	US3	Not done
AC9	Prepare the overall project plan (chapter 1) for the first submission	US4	Verified by group
AC10	Prepare the chapter about project management (chapter 2) for the first submission	US4	Verified by group
AC11	Prepare the pre-study (chapter 3) for the first submission	US4	Verified by group
AC12	Prepare the requirement specification document (chapter 4) for the first submission	US4	Verified by group

that we probably had chosen to much work for this sprint. We listened to them and splitted a task concerning the camera as another user story. Then we prioritized US3 as a story that only should be done if we got time after the other tasks in this sprint.

7.3 Testing

In this section we discuss the tests we have ran and how we verified the acceptance criteria of the sprint.

In this sprint we used unit testing to ensure that code behaves as intended and holds an acceptable quality level, we also performed some minor "system tests" and acceptance testing to the acceptance criterias made for this sprint. We did this to ensure that the parts of the code we were unable to test with user tests behaved correctly, as well as that

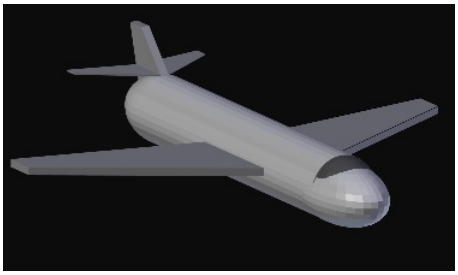


Figure 7.1: Rendering of the 3D plane model

the acceptance criteria were fulfilled.

During the sprint we were using the methodology of test driven development. This ensured that our classes were written with testability in mind, which in turn made the code more maintainable and easier to read. Test driven development does however have its shortcomings, especially when it comes to graphical user interface testing. To test the parts of the code which involved changes in the graphical user interface, we relied simply on manual verification of each of the functions.

At the end of the sprint we wanted to verify which of the goals we had managed to fulfill, as shown in Table 7.2. This was done by manual verification. Acceptance criteria 1 and 2 were performed by making sure that everyone was able to utilize the built in unit test functionality of Visual Studio. Criteria 3 and 5 were tested by running a demo where we made sure that our planes behaved as planned. Criteria 7 was done after we had documented our reasoning thoroughly. This can be seen in subsection 3.5.3. We were unable to complete acceptance criteria 8, and without the Kinect device we were also unable to fulfill the requirement of acceptance criteria 6 as well.

7.4 Results

Here we describe the results of what we did on the programming part the first sprint. We managed to set the XNA project up and running, and got some planes in "the air". These where first made as a 3D model of the plane with blender. The result is showed in Figure 7.1.These planes all have different positions and velocity. The user is able to select a single plane by typing in the name of the plane and then pressing enter. Then the user can set the direction the plane is going. This selection and setting direction process will later be altered to be voice controlled.

Figure 7.2 and Figure 7.3 are screenshots from our running system. From Figure 7.2 we can see airplanes with different position, rotation and pitch, while we from Figure 7.3 can see that the planes have moved in the direction they are going. Some have moved further than others because of different speeds.

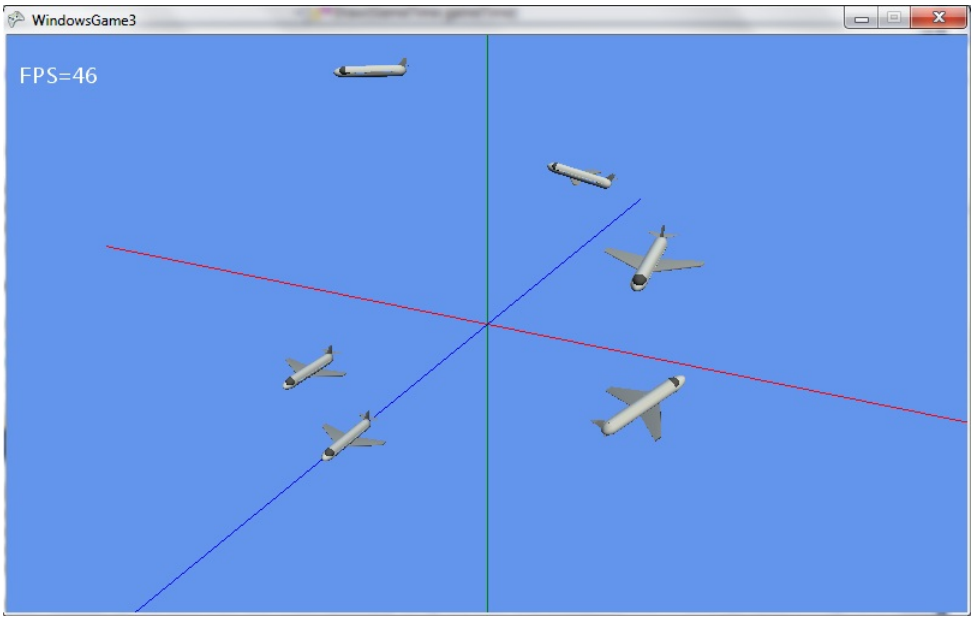


Figure 7.2: Screenshot showing planes in the airspace

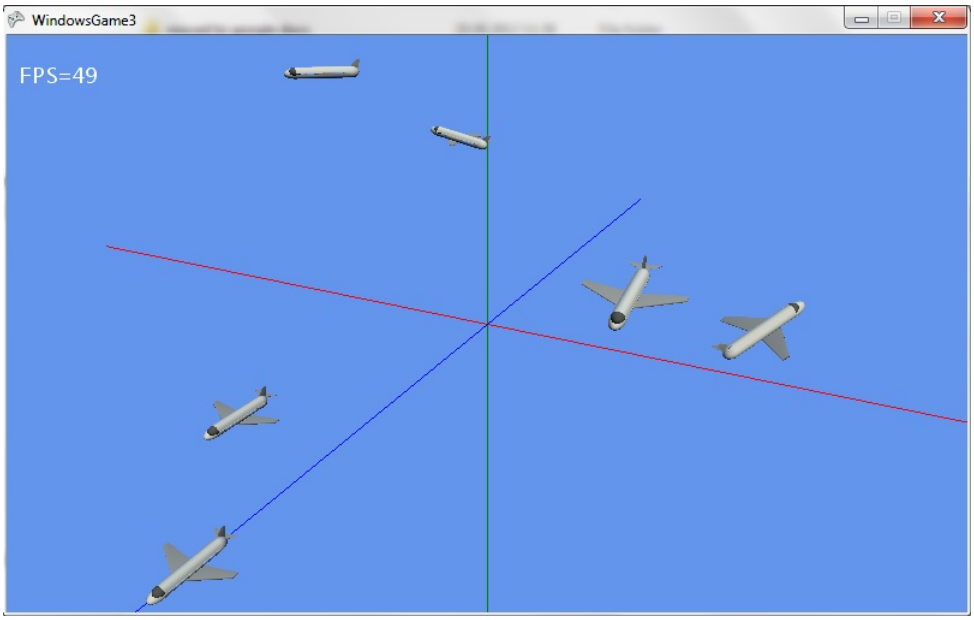


Figure 7.3: Screenshot showing the planes from Figure 7.2 after some time, with a new position

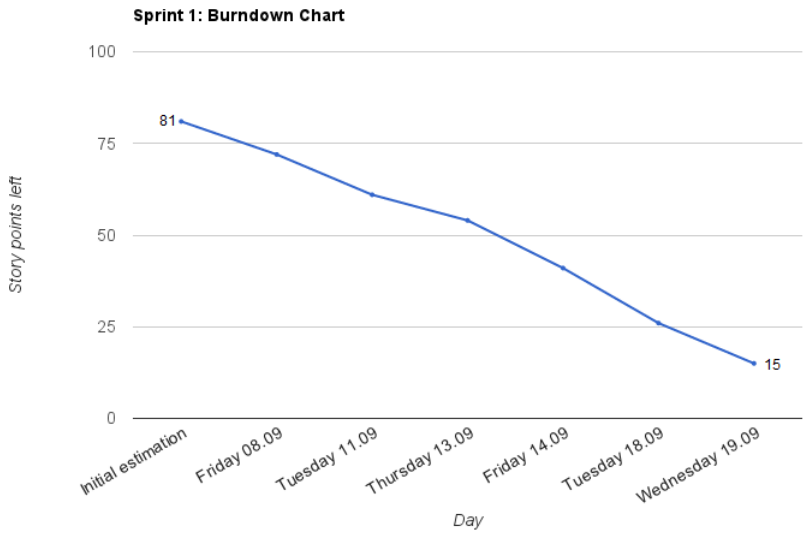


Figure 7.4: The burndown chart for the first sprint

7.5 Sprint Retrospective

The burndown chart from the sprint is shown in Figure 7.4.

We can see the number of story points left goes downs nice and steady. Even though we did not get done with all the task in the sprint backlog, we are happy with the sprint because we reached the high prioritized goals, and it was our first sprint so the estimations would not be perfect. We used 250 hours on this sprint, where 147 was on the engineering tasks, and the rest on other activities like meetings, lectures, planning and project management. In the 147 hours we managed to do 66 story points, so we used 2 hours and 14 minutes. We can use this for estimation reference in the future. There is a table in section G.1 showing each engineering task for this sprint, the estimation, the story points left and the hours used.

There were several reasons why we did not get time to do all the task under US3. We overestimated the work we could do and had a lot to do in other courses. After the meeting with the advisor at 13.09 we came to an agreement that the documentation was lacking, and should be higher prioritized in the upcoming sprint. We came up with a solution to this, which was to add a dedicated user story for documentation.

We also used more time on programming during this sprint than expected. This had two reasons. First of all, some of the engineering tasks were not broken down into small enough pieces. Second of all, some of the engineering tasks were not thought through well enough. We decided to solve this by using a bit more time in the sprint planning, making

sure we broke the user stories into small enough pieces, and that every task was necessary and meaningful.

Another problem we stumbled across during the first sprint was the handover of the source control repository. We got access one week later than the date agreed upon, and had to use git and github in the meantime. Since this is a one time event, we do not have a solution if it happens again, but we will keep in mind when planning other sprints that tasks could become delayed, and have a backup plan if there are other tasks depending on this delayed task. The Kinect device was also received later than expected, which meant we were unable to finish the tasks regarding Kinect programming.

Most of the team also caught a cold during the sprint, and even though most could work from home, communication were limited and performance a bit worse. If this happens in the future, we will stick with the risk plan in Appendix C, which says that the work should be distributed amongst the rest of the group and the ill member should be kept up to date.

In the end we still managed to do the most important task, and were quite pleased with that.

We had a retrospective meeting after the sprint review meeting the 20th of September. Here is a summary of the most important feedback we gave each other.

Like we described in the paragraph above, we should start to divide the engineering task better at the beginning of the sprint. Updating hours, story points left and written documents should be done right away online. The test coverage should be improved. We should plan more hours to do group work, but also be better at having small breaks more often. Most importantly we need to speak more in English to include the whole group in a better way. We should continue having positive attitude towards other group members, and being help friendly. It it easy to ask for help from other group members. The daily scrum meeting was useful so we will continue with that as well. We should stop coming late to group work sessions, and limit the times we are assigned to more than one task at the same time.

So the plan is to focus on getting the topics discussed here better to next time, and then have a new sprint retrospective meeting, and hopefully we have improved on some of the topics.

Chapter 8

Sprint 2

This was our second sprint. The group decided to have Eirik as Scrum Master for this sprint, as he was the second person in our group with Scrum Master experience from previous projects.

This chapter is divided into the following parts. The sprint breakdown with all the meetings and important dates. The sprint planning with our overall plan for this sprint, and the user stories we are going to do. The chapter about testing, where we discuss how we tested the tasks and user stories we finished. The results chapter where write about what we finished, with some screenshots of our system. Finally, the sprint retrospective with the hours we have used on each task, the burndown chart and reflection.

8.1 Sprint Breakdown

The sprint started on the 20th September with a planning meeting at 09:30 with the team and the customer on Skype. Later the same day we had a supervisor meeting at 13:00. We planned to have a daily scrum meeting for about 10 minutes with the team members, and we were going to work together on Tuesday, Thursday, half of Friday and half of Wednesday. A supervisor meeting was to be held at the 27th of September. At the end of the sprint, the 4th of October, we should have a sprint demo/review meeting with the customer at 09:30. After this meeting we were going to end the sprint with a sprint retrospective meeting at 10:30.

8.2 Sprint Planning

On the planning meeting with the customer we decided which user stories we were going to work on. Making a green ground was higher prioritized by the customer than we thought, and was the first implementation task this sprint. We asked them for acceptance criteria for the implementation side, and made the ones for documentation our self. Our plan for

Table 8.1: The current user stories in the sprint backlog

ID	User story	Overall priority	Priority this sprint
US4	As a student, I need to document the work thoroughly, so that we can write a good report of our project.	High	High
US5	As an air traffic controller, I want to be able to see the ground.	High	High
US3	As a programmer, I need to learn the Kinect framework, so that I can implement the system to support Kinect.	High	Medium

this sprint was to do a lot of documentation, finishing up the documentation from the first sprint, and working on the recommendation the supervisor had given us about changes in the first chapters. This included writing it in LaTeX, which should take a lot of time. For the implementation we were going to make the simple flat green ground. Then we will make a small demo with basic gestures in Kinect, which is a huge task because we did not have Kinect in the prestudy phase. So we needed to learn to use it, together with its framework.

Table 8.1 shows the user stories we were going to work on this sprint. They were the highest prioritized user stories at that point in time.

First we divided the user stories into smaller engineering tasks (ET) and then we used planning poker to assign them story point. With better knowledge of our working capacity and speed, we should get better estimates than in the first sprint. We have planned 74 story points this sprint. More details in the sprint backlog with engineering tasks and there estimations are shown in section G.2 in the Appendix.

8.3 Testing

In this section we discuss the tests we have run and how we verified the acceptance criteria of the sprint. The acceptance criteria are shown in Table 8.2.

The tests performed during this sprint were unit testing to ensure that code behaved as intended and had an acceptable quality level, and acceptance tests and integration tests related to the acceptance criteria for the sprint and the finished part of the system to ensure that the parts of the code we were unable to test with user tests behaved correctly.

We did not do much change in the main program this sprint. We did find one bug, and

Table 8.2: Acceptance criteria for the second sprint and their evaluation

ID	**Acceptance criteria**	**User story**	**Verified Finished**
AC12	Make the changes in chapter 1- 4 recommended by the supervisor	US4	Verified by group
AC13	The documentation of sprint 1 are finished and delivered to the supervisor	US4	Verified by group
AC14	The text written about sprint 2 are finished and delivered to the customer and supervisor	US4	Verified by goup
AC9	Having a working Kinect demo with some basic gestures	US3	Verified by customer
AC10	Having a flat, green ground	US5	Verified by customer
AC11	Get a brief overview of the speech feature of the Kinect SDK	US3	Verified by customer

made sure to add unit tests to ensure that it works now, and that we can test it easily in the future.

We could not test acceptance criteria 9 automaticly, because that would require us to add mocking for Kinect. As discussed in subsection 3.5.4 this would take too much time to set up. Thus we need to this manually. We tested that the user could both rotate and move the 3D camera model. This is showed in Figure 8.2 and Figure 8.3.

Acceptance criteria 10 can be seen as finished from Figure 8.1. We also made sure that the airplanes is not displayed if they are under the ground level. Having any more advanced collision detections between the airplanes and the ground was not high prioritized for this project.

Acceptance criteria 11 was done after we had documented our research thoroughly, as seen in subsection 3.5.6. We also made a small program for this which could recognise a few words. This was enough to get a brief overview.

8.4 Results

Here we describe the results of what we did on the programming part this sprint. We managed to create a green ground. Even though it was an high prioritized user story, we should not use time to make it look very good. For now the customer just wanted a simple, flat, green ground. We still have the axis showing for now. We also changed the title of the program to Speech and motion driven UX. These two changes was the only visual change

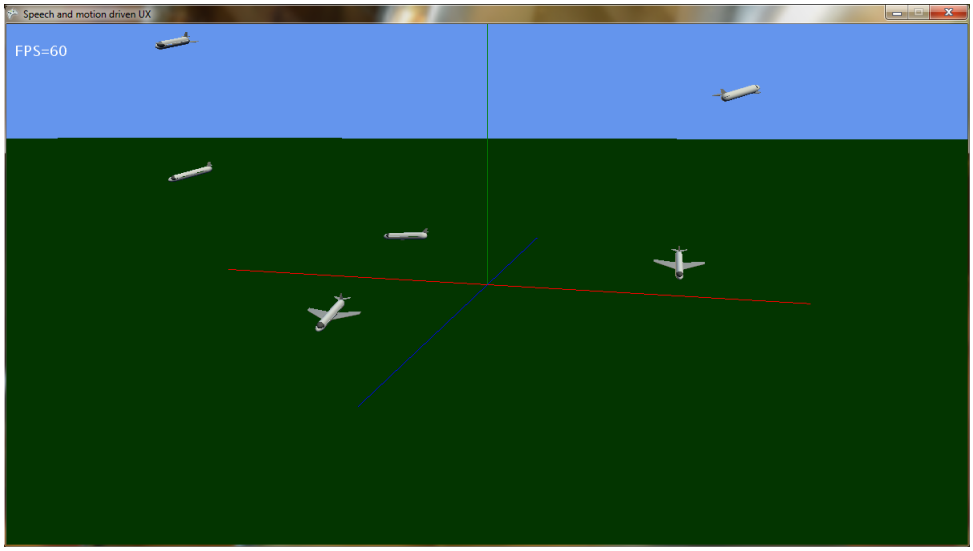


Figure 8.1: Screenshot of our project with green ground.

we made to the main software program this sprint. Having the airplanes so close to each other is just for testing purposes right now.

From Figure 8.1 we can see the green ground, and the new title of the program in the upper left corner.

The customer also wanted a Kinect demo, where we should show what we had found during the Kinect research. This should be done by having a simple 3D model on screen that we were able to move and rotate using Kinect and hand motion. We made a simple looking 3D camera model because it is the camera that should be controlled in the main program in later stages.

From Figure 8.2 we can see our Kinect Demo. The yellow and green circles represents the left and right hand of the user. As described in the screenshot, moving your hand over the horizontal line starts the gesture.

From Figure 8.3 we can see the camera has been rotated to the right, and moved a little into the screen.

8.5 Sprint Retrospective

Looking at Figure 8.4, the burndown chart, we can see that the number of story points left going down on a steeper curve than in the first sprint. More details can be found in the sprint backlog in section G.2. This sprint we used 246 hours working, where 147 was on the engineering tasks, and the rest on other activities like meetings, lectures, planning and project management. In the 147 hours we managed to do 84 story points, so we used

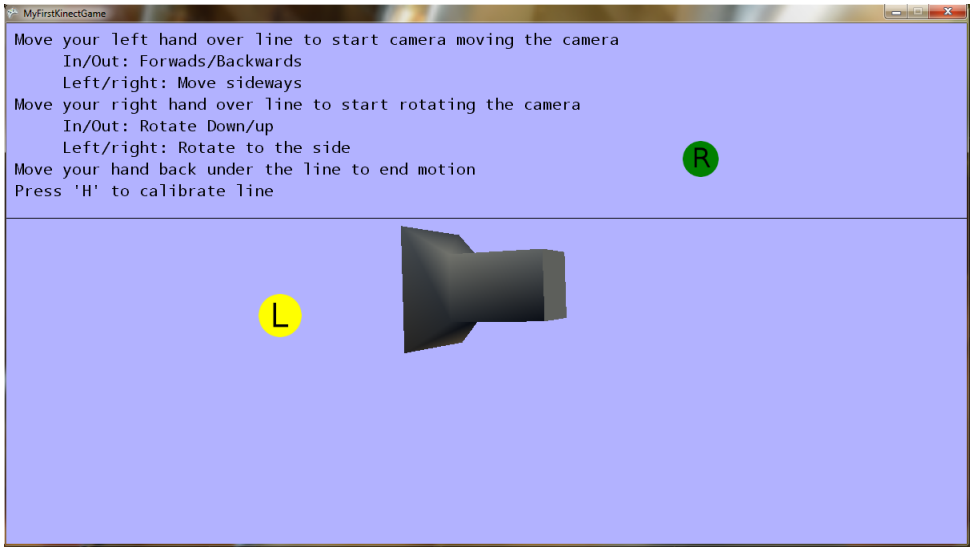


Figure 8.2: Screenshot of the Kinect demonstration system

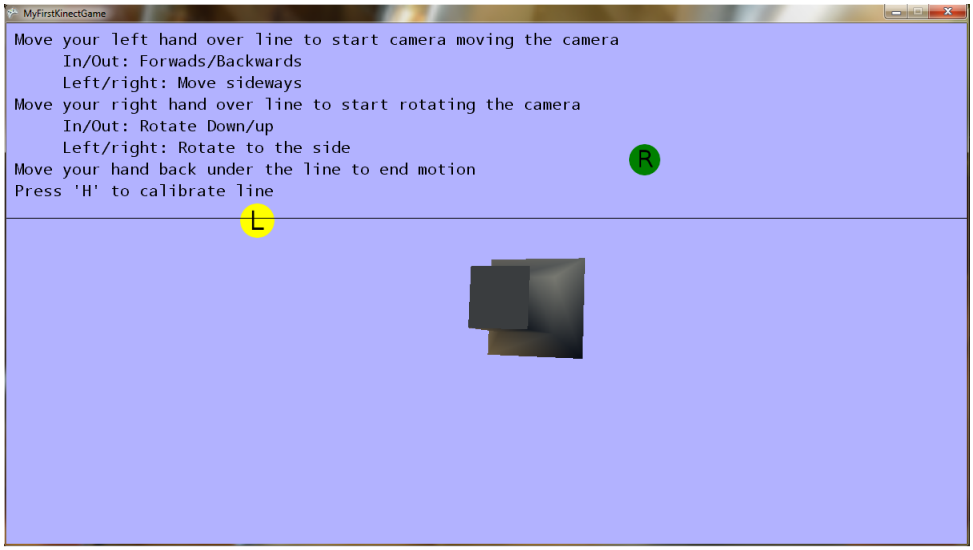


Figure 8.3: Screenshot of the Kinect demonstration system after the user has moved his hands

on average 1 hour and 3 quarters on each story point, which is slightly faster than in the first sprint.

At the supervisor meeting the 27th of September we got some pointers for changes we needed to do in the documentation. At that point we were down to 17 story points left,

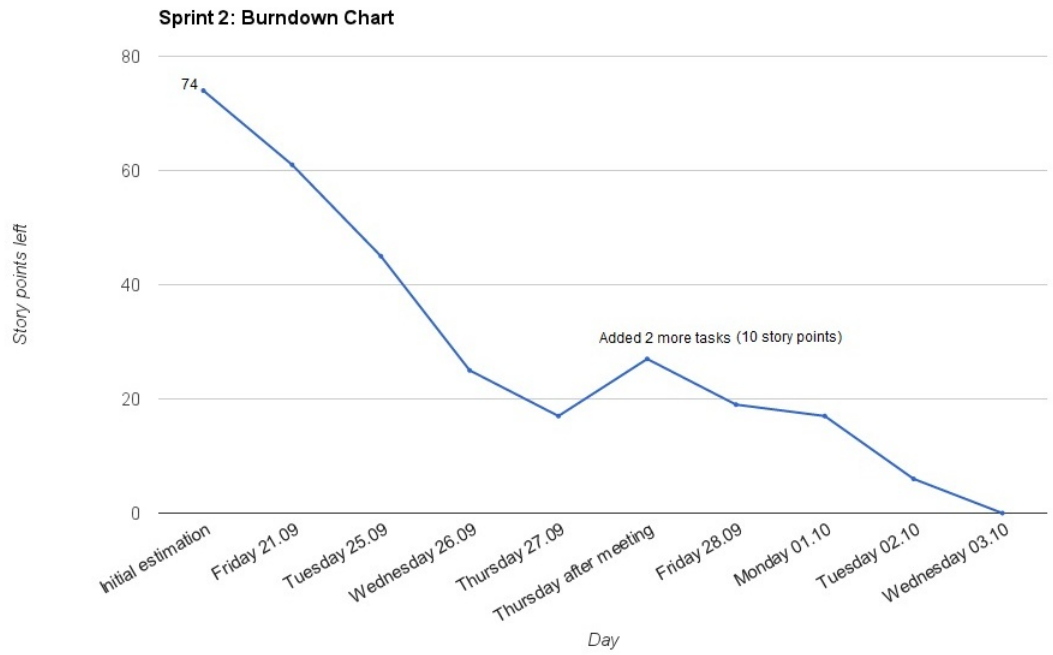


Figure 8.4: The burndown chart from the second sprint

and was ahead of schedule with one week left. We thought we could manage to do more tasks, so we added ETs for making changes in the already existing documentation (ET45) and proofreading (ET46). For later sprints we should consider to add tasks for changes in the documentation from the the beginning. This is because the supervisor will most likely find more areas of improvement each week, so we should estimate for that as well.

There are also several reasons why the curve was steeper this sprint. Like described we finished an average story point faster than the first sprint. All the group members had recovered from the cold that many of us had the first sprint, and the beginning of the sprint Johanne was working overtime because she wanted to save up some time and leave for a long weekend in London from Thursday 27.09. We used less time on project management this sprint, having the TFS server and online Sprint boards already running. We also worked better on dividing the ETs into smaller, more well defined tasks in this sprint. Finally some of the ETs was faster to solve than we had estimated.

One of those were ET27 (Write sprint introduction, breakdown and planning for sprint 2). Here we estimated 3 story points, because it took a lot of time the first sprint. We used some hours the first sprint making choices about structuring and finding topics to write about. This time we could rely on the decisions we made the first time and copy much of the structure. This time we only used one hour on the task.

Another was some of the tasks concerning US3 (As a programmer, I need to learn the

Kinect framework, so that I can implement the system to support Kinect) where we also used less time that estimated. We estimated 33 story points for the tasks to solving US3. Using the numbers from the first sprint we would use about 2 hours to solve each story. So as an estimation we would use 66 hours on these tasks, but we ended up using 48. Especially setting up the environment between XNA and Kinect was more straightforward than we thought. Also making the demo took less time than expected because much of the functionality was already implemented in ET39 (Learn to recognise basic hand motions/gestures, and document the research). These tasks were hard to estimate good because we had not worked with Kinect before, and did not have the Kinect during the prestudy phase.

We had a sprint retrospective meeting after the demo/review meeting with the customer at the 4th of October. In the last sprint retrospective we found some areas where we could improve as a team. This time we were glad to see that the whole team agreed that we had made progress. Several of the subjects we said we should start with the last meeting we now say we should continue doing.

Examples include updating the hours used on project document each day, having more group work hours, dividing up engineering tasks in a good way, and having more than one person assigned to importing documents to LaTeX. We have also become much better at including the whole group by always speaking in English, and will continue doing that. We also managed to stop coming late to group work sessions and stop being assigned to too many task at the same time, which we will continue with. Like in the other sprints we agreed to continue having positive attitude towards other group members, and being help friendly. It it still easy to ask for help from other group members.

Although the list was much shorter than the first time, we still found some areas we could improve. We wanted to have more small breaks during working hours. Another thing we should start doing was more pair programming so that we all get a better overview of the code together. We also wanted to have a social evening together with the whole team for further team building and enjoyment. In the end we did not find anything we needed to stop doing.

Chapter 9

Sprint 3

This was our third sprint. The group decided to have Johanne as Scrum Master for this sprint. She was motivated to get experience with that role.

This chapter is divided into the following parts. The sprint breakdown with all the meetings and important dates. The sprint planning with our overall plan for this sprint and the user stories we are going to do. The section about testing, where we discuss how we tested the tasks and user stories we finished. The results section where write about what we finished, with some screenshots of our system. Finally, the sprint retrospective with the hours we have used on each storypoint, the burndown chart and reflections.

9.1 Sprint Breakdown

The sprint started on the 4h of October with a planning meeting at 09:30 with the team and the customer on Skype. Later the same day we had a supervisor meeting at 12:00. We planned to have a daily Scrum meeting for about 10 minutes with the team members, and we were going to work together on Tuesday, Thursday, half of Friday and half of Wednesday. A supervisor meeting was to be held at the 11th of October. On Sunday 14th of October we needed to have the documentation ready for the pre-delivery. At the end of the sprint, the 18th of October, we should have a sprint demo/review meeting with the customer at 09:30. This meeting were postponed a week, so the demonstration of the product did not take place before 25th of October at 11:30. Since someone had to go away the 18th of October, we decided to have the sprint retrospective meeting at Tuesday 23rd of October at 12:00.

9.2 Sprint Planning

On the planning meeting with the customer we decided which user stories we were going to work on. We told the customer that we had a delivery deadline for the pre-delivery of the

Table 9.1: The current user stories in the sprint backlog

ID	User story	Overall priority	Priority this sprint
US4	As a student, I need to document the work thoroughly, so that we can write a good report of our project.	High	High
US6	As an air traffic controller, I want to be able to rotate and move the camera, so that I can get a better overview of where the planes are relative to one another.	High	High
US7	As an air traffic controller, I want to use hand gestures to change the camera direction and position, so that I can control the system without keyboard.	High	Medium

documentation during the sprint, and he agreed that we should focus on the documentation the first week of the sprint, and in the second week focus on implementation tasks.

For the documentation we wrote more in the overall system design and the test chapter, wrote the sprint 3 chapter and improved the documentation we had already written, and imported everything to LaTeX.

The customer were happy with our progress and our Kinect prototype, and wanted us to start implementing some of the features from the prototype in the main project. Before the Kinect implementation we needed to finish the Camera class, which is the part of the system that the Kinect was going to control. This camera would be used as a viewpoint, the location the user would see from when looking in the 3D environment. After this was done we would implement it so that the user could use the Kinect to control the camera. We asked them for acceptance criteria for the implementation side, and made the ones for documentation our self.

Table 9.1 shows the user stories we were going to work on this sprint. These are our highest prioritized user stories.

9.3 Testing

In this section we discuss the tests we have conducted and how we verified the acceptance criteria for the sprint. The acceptance criteria are shown in Table 9.2.

In this sprint we conducted two forms of test: Unit tests to ensure that small units of

Table 9.2: Acceptance criteria for the third sprint and their evaluation

ID	**Acceptance criteria**	**User story**	**Verified Finished**
AC15	Get documentation ready for preliminary delivery	US4	Verified by group
AC16	The camera is able to zoom in and out	US6	Verified by customer
AC17	The camera is able to rotate right and left	US6	Verified by customer
AC18	The camera can be rotated right and left using the Kinect	US7	Verified by customer
AC19	The camera is controlled by use of the kinect to zoom in and out	US7	Verified by customer

the program behaved as intended, and gave the correct outputs/ results when the program were ran, and integration tests for the parts of the system that were difficult to test by unit testing. The integration tests concerned integration between motion and the model.

Unit tests were written regularly for all the new methods where this were possible. The integration tests were conducted during the implementation, as well as at the end of the sprint.

9.4 Results

Here we describe the results of what we did on the documentation and the programming part of this sprint.

The milestone regarding delivery of a preliminary version of our report was reached in this sprint. The preliminary version of the documentation was finished, up to the seven first chapters of the report. Also, a preliminary version of the architecture were drafted in section 5.6, using the 4+1 view model. The report were proofread and delivered on time. All the available hours of the first week were used on the report.

The second week were used for coding. The focus was on making a "camera," letting the user rotate, move and zoom in the model. At our first implementation, the camera was controlled by the number pad keys. Next, we added support for gestures using Kinect. This was based on the work we did in the previous sprint.

During the work with controlling the camera using motion, we realised that we had no good way to notify the user of how his motions affected the model. In the camera test program we made last sprint, we had solved this by showing the users hands as dots on

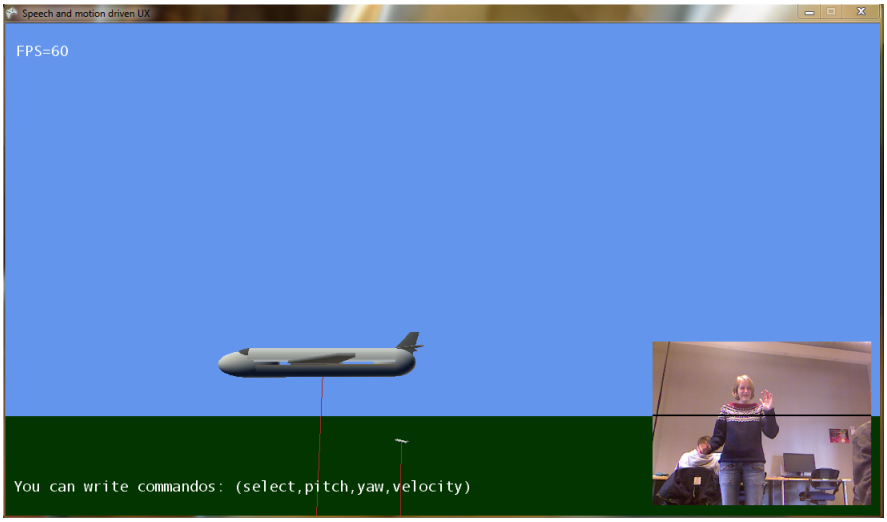


Figure 9.1: Screenshot showing the window with the video of the user at the bottom right corner

the screen. But we felt that this would not look good in the model of the airspace. As a result, we decided to create a dedicated view, displaying a video of the user and the line representing the height where motions are captured. This display should be shown in a small square in one of the corners of our system. A screenshot showing our program running with the new view can be found in Figure 9.1.

9.5 Sprint Retrospective

Looking at Figure 9.2, we can see that the curve goes down at a steady rate, starting a bit slower than the average rate. The reason for the slow start was that not all of us were able to work on Friday the 5th of October. In this sprint we used 228 hours working, where 158 were on engineering tasks. The reason that the hours used on engineering task are higher than in the previous weeks, is that we managed to express the most of the documentation that had to be done as engineering tasks. Also, there was no lectures during this sprint, and we used less time on project management. In total, we have worked 22 hours less than expected. The reason for this is that not everybody had time to work at the end of this sprint, and they were going to catch up with their hours in the weekend after this sprint was finished.

We started out with an estimated workload of 86 story points, and managed to do all the story points during this sprint. On average, we used aproximately 1 hour 50 minutes per story point. From the sprint backlog in section G.3, we can see that the most of the tasks had an OK workload estimation. A big exception from this is ET55, make changes

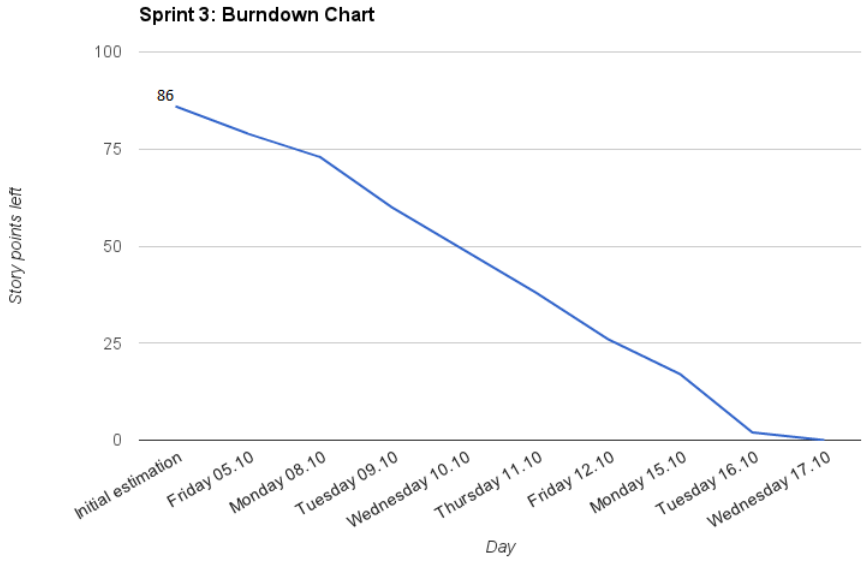


Figure 9.2: The burndown chart from the third sprint

to existing documentation. This task were estimated as an 8, but it was the task that we used the most time on, 33 hours, giving an average of around 4 hours 8 minutes per story point.

The retrospective meeting was held at 23rd of October, some days after the sprint was finished. The reason for not having it the same day as the sprint ended was that not all of us were present at that time. The results of the retrospective meeting were divided in three categories, where the category containing things we should stop doing were empty. The other two categories were things we should continue doing and things we should start doing.

The start section mentioned two things: to make coding conventions for variable names in the code and to gather for a social evening. Everybody were interested in having a social event, so we scheduled a taco and Kinect evening at the 28th of October. We also decided on some new coding conventions: global variables should start with a capital letter, while local should start with a lowercase letter.

The continue section contained a lot of the same as last time. We should continue speaking English, being positive and help friendly towards each other, using pair programming and working together at school. Other things that were mentioned were that we should continue dividing the work into smaller task, continue to write the hours worked in a shared document and continue coming on time. We were happy with all of this being mentioned, because this shows that we were all satisfied with the way we are working

together and we do not need to conduct major changes to our group sessions.

Chapter 10

Sprint 4

This was the fourth sprint in our project. This time, Mahboobeh was going to be the Scrum master.

This chapter is divided into the following sections. The sprint breakdown with all the meetings and important dates. The Sprint planning with our overall plan for this sprint and the user stories we were going to do. The section about testing, where we discuss how we tested the tasks and user stories we finished. Next, The result section where we write about what we finished, with some screenshots of our system, and finally, the sprint retrospective with the hours we have used on each story point, the burndown chart and reflections.

10.1 Sprint Breakdown

The Sprint started on the 18th of October with a planning meeting with the group. The customer were not able to participate, so he had sent us an email with which engineering tasks he wanted us to focus on, and acceptance criteria for them. Later the same day we had a supervisor meeting at 13:00. We planned to have a daily Scrum meeting for about 10 minutes with the team members, and work together on Tuesday, Thursday, half of Friday and half of Wednesday. The 22nd of October a lecture on technical writing were scheduled, which all of the group members should attend. At Thursday 25th of October, a meeting with the customer should be held at 11:00, where we should demonstrate the work we did in the last sprint. Later the same day, at 13:00, there should be a meeting with our supervisor. The sprint should end with a demo/review meeting with the customer at 09:30 at the 1st of November, and a retrospective meeting with the group at 10:30 the same day.

10.2 Sprint Planning

Since the customer was not able to attend the sprint planning meeting, he sent us an email with the user stories he wanted us to focus on, and acceptance criteria for them. He also wanted us to change the prioritizing for some of the user stories in the product backlog (US14 and US15), and also to add a new user story, US17. We agreed with him in the changes he proposed, and also added the new user story. We have already established this as a risk with a solution, shown in Appendix C in Table C.12.

The main focus in this sprint was to integrate the voice with our airplane model, which led to the overall goal: a working model with both motion control for the camera and voice control for the airplanes. Further, it should be possible to select a single airplane and displaying information about it. Also, a possibility of displaying the names and coordinates of all the airplanes in the model was preferred. This should all be achieved using voice.

A lecture in technical writing was scheduled in the first week, which we all should attend. We wanted to continue our focus on documentation in this sprint, and try to improve our report even further, hopefully with some help from the technical writing course. Also, missing sections and paragraphs must be added.

Table 10.1 shows the user stories we were going to work with in this sprint.

10.3 Testing

This section contains information about the tests we conducted during this sprint. The acceptance criteria for this sprint is shown in Table 10.2.

In this sprint we used both unit tests and usability testing. Unit testing were mostly used during the coding of the speech recognition part. This was done by implementing that the program should recognize a word or a sentence, and then test if this happened by saying the word while the program ran, and see if it was interpreted correctly.

Also, we decided to start with testing of the system by users in this sprint. Since we have all our major functionality in place in our program, we figured this as a good time to start testing. The test were conducted the 31st of October, and two persons participated. None of them had seen our system before, but they both had some minor previous experience with Kinect.

The results from the usability tests can be found in subsection 12.1.5. The usability testing gave us some small things that we should improve, like displaying feedback on the users input. This was done by adding a text line at the bottom of the screen, showing the interpreting of the users voice input. Also, the text box displaying the possible speech commands were changed a bit, changing the order some of the commands were displayed.

The users struggled a bit with calibration of the line above where motions are recorded. Therefore, we decided to remove this line completely, setting the lowest point where motions is registered to shoulder height.

Table 10.1: The current user stories in the sprint backlog

ID	**User story**	**Overall priority**	**Priority this sprint**
US4	As a student, I need to document the work thoroughly, so that we can write a good report of our project.	High	High
US8	As an air traffic controller, I want to give speech commands to select an airplane, so that I can give it more commands later.	High	High
US9	As an air traffic controller, I want to give speech commands to an already selected airplane, so that I can alter their direction of movement.	High	High
US17	As an airplane, I should display a menu with the airplane's data and additional commands when selected.	Medium	Medium
US10	As an air traffic controller, I want to see the names of the different planes, so that I can distinguish them in a easy way.	Medium	Medium
US11	As an air traffic controller, I want to see the airplanes' coordinates, so that I get a better feeling where the airplanes are.	Medium	Low
US14	As an air traffic controller, I want to be able to move the camera close to the selected plane with a speech command, so that I can easily move towards a plane.	Low	Low
US18	As an air traffic controller, I want to see some terrain, making it easier to see how the planes are positioned relative to each other.	Low	Low

Table 10.2: Acceptance criteria for the fourth sprint and their evaluation

ID	**Acceptance criteria**	**User story**	**Verified Finished**
AC20	The documents should be changed according to the reviews from the course staff/ censor.	US4	Verified by group
AC28	The chapter about sprint 4 should be finished (without sprint retrospective meeting).	US4	Verified by group
AC21	Given the command "select planename" the given plane is shown in the model as selected.	US8	Verified by customer
AC22	When selected, additional data should be viewable, like altitude, speed and remaining fuel.	US17	Verified by customer
AC23	When selected, the available speech commands are displayed.	US9	Verified by customer
AC24	Given the command "Alter heading direction" where direction is a 360 degree value, the selected plane would change its course accordingly.	US17	Verified by customer
AC25	When a airplane changes its direction, this is done gradually.	US10	Verified by customer
AC26	The names of the planes should be displayed along with the plane. It should be possible to toggle names on and off with the command "Toggle name".	US17	Verified by customer
AC27	The coordinates of the planes should be displayed close to the plane. It should be possible to toggle coordinates on and off with the command "Toggle coordinates".	US11	Verified by customer
AC28	One can get a clear view of the selected airplane when the command "go to" is given.	US14	Verified by customer
AC29	The terrain is in 3D with some mountains, there is space for the planes to navigate and the altitude is noticeable.	US18	Verified by customer

10.4 Results

This section describes what we did on programming and documentation in this sprint.

In this sprint we have focused on integrating voice recognition with our program. The voice recognition is used for selecting a single plane, controlling a plane when it is selected and to toggle the displaying of all the airplane's coordinates and names.

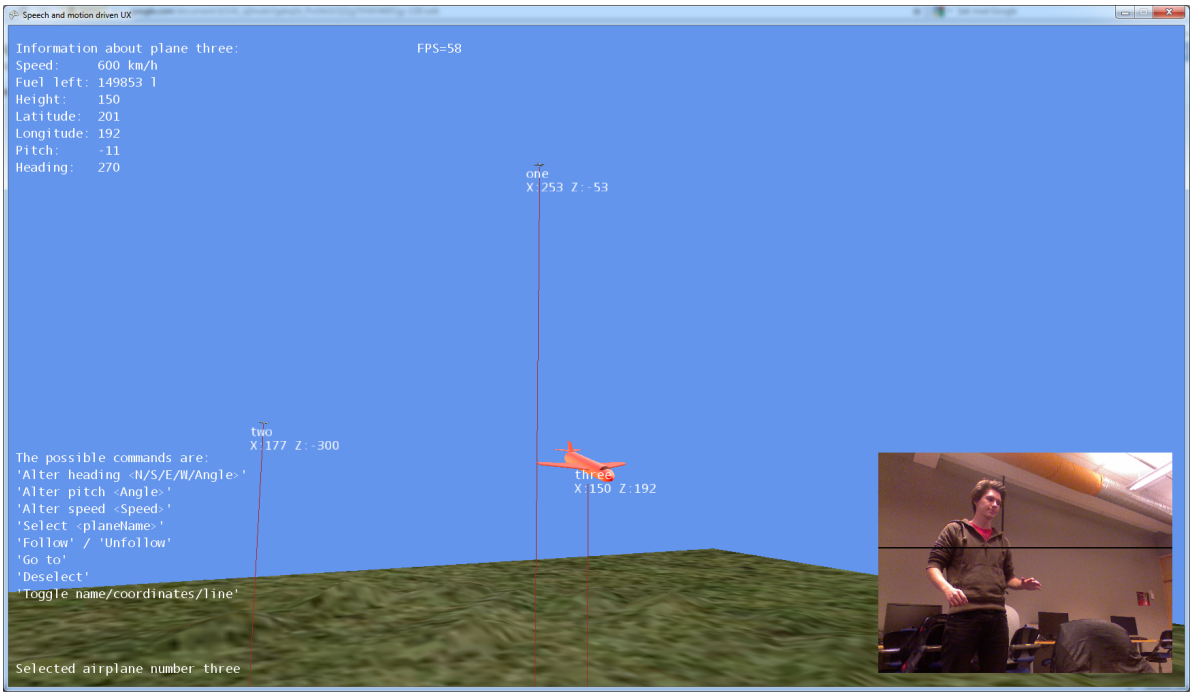


Figure 10.1: Screenshot showing a selected airplane (red) with its information and the possible speech commands

To be able to use voice for all this, we have added the ability to set an airplane as selected (shown as red), and checks so that only one airplane is selected at all time. Also, we have added a textbox to screen showing the selected airplane's information, a textbox with possible voice commands and the ability to display text above all of the airplanes. A screenshot showing a selected plane, its displayed information and the possible comands is shown in Figure 10.1.

The other things we have added were a fuel variable, showing the selected airplane's remaining fuel, changes to the turning of the plane, making it turn more realistic instead of just heading in the new direction instantly and acceleration when the airplane changes its speed.

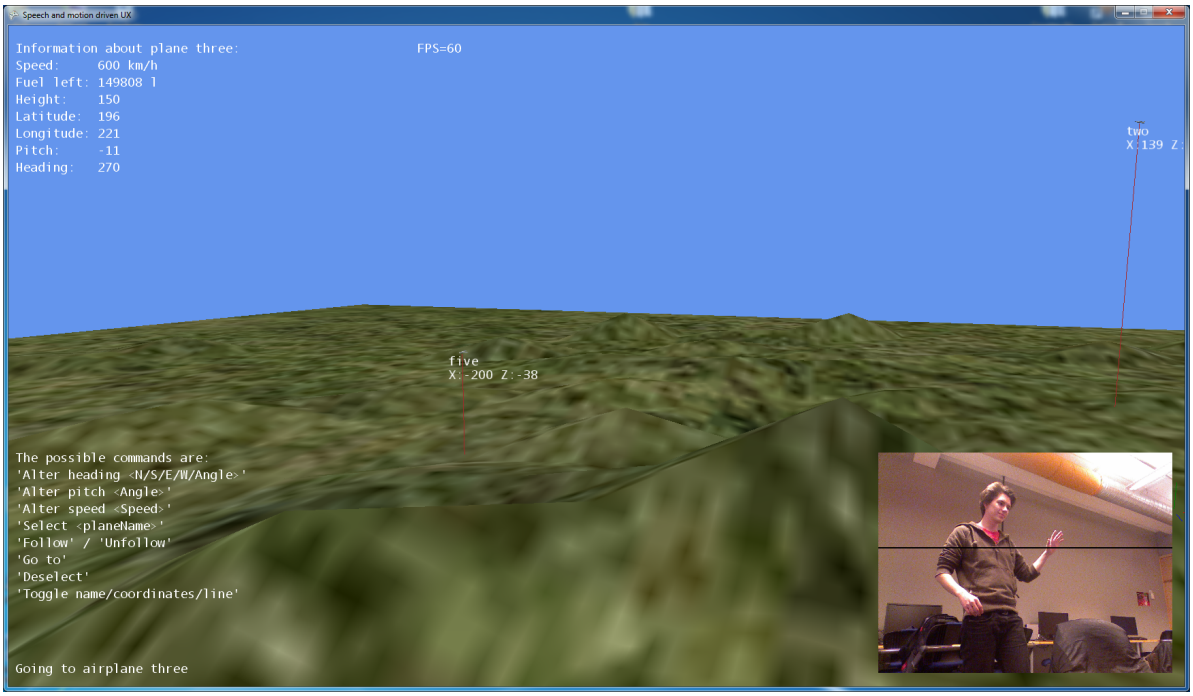


Figure 10.2: Screenshot showing the terrain in the model

The review meeting with the customer were postponed from last sprint, so the meeting was held in the middle of this sprint instead, at 25th of October. At this time, we had finished the most of the programming for this sprint, and we had not received any reviews on the preliminary delivery, so the customer came up with some new tasks he wanted us to focus on. One of them was to do something with the ground, to make it easier to see where the planes were positioned relative to each other. The other task was to make it possible to zoom in on the selected airplane using a single command.

Both of these tasks were solved during this sprint. A new model of the ground was created, shown in Figure 10.2. with textures and some height differences on the surface. Making it look more real. Also, the possibility to zoom in on a given airplane using the voice command go to and the command follow for following a plane.

The speech commands we have at the end of this sprint are shown in Table 10.3.

For the documentation part we have worked a lot with getting the references and our bibliography correct, reading through the report and making sure that all the places where we have used sources, contains references to those.

We did also receive a list of the changes we should do to the documents after the preliminary delivery. We have conducted all the suggested changes, except for some we

Table 10.3: The available speech commands

Command	**When**	**Result**
Calibrate	Always	The line where movement is registered is moved to level of the user's right hand
Toggle names	Always	The names of the airplanes are displayed/ hidden
Toggle coordinates	Always	The coordinates of the airplanes are displayed/ hidden
Toggle lines	Always	The lines below the airplanes are displayed/ hidden
Select plane <planeName>	Always	The airplane with name planeName is shown as selected
Alter heading <north, south, east, west, degrees>	When an airplane is selected	The heading of the selected airplane is changed
Alter speed <speed>	When an airplane is selected	The speed of the selected airplane is changed
Alter pitch <degrees>(-15 to 15)	After an airplane is selected	The pitch of the selected airplane is changed
Go to	When an airplane is selected	The camera moves to the selected airplane
Follow	When an airplane is selected	The camera follows the selected airplane
Unfollow	After the command follow has been issued	The camera stops following the selected airplane
Deselect	When an airplane is selected	The selected airplane is unselected

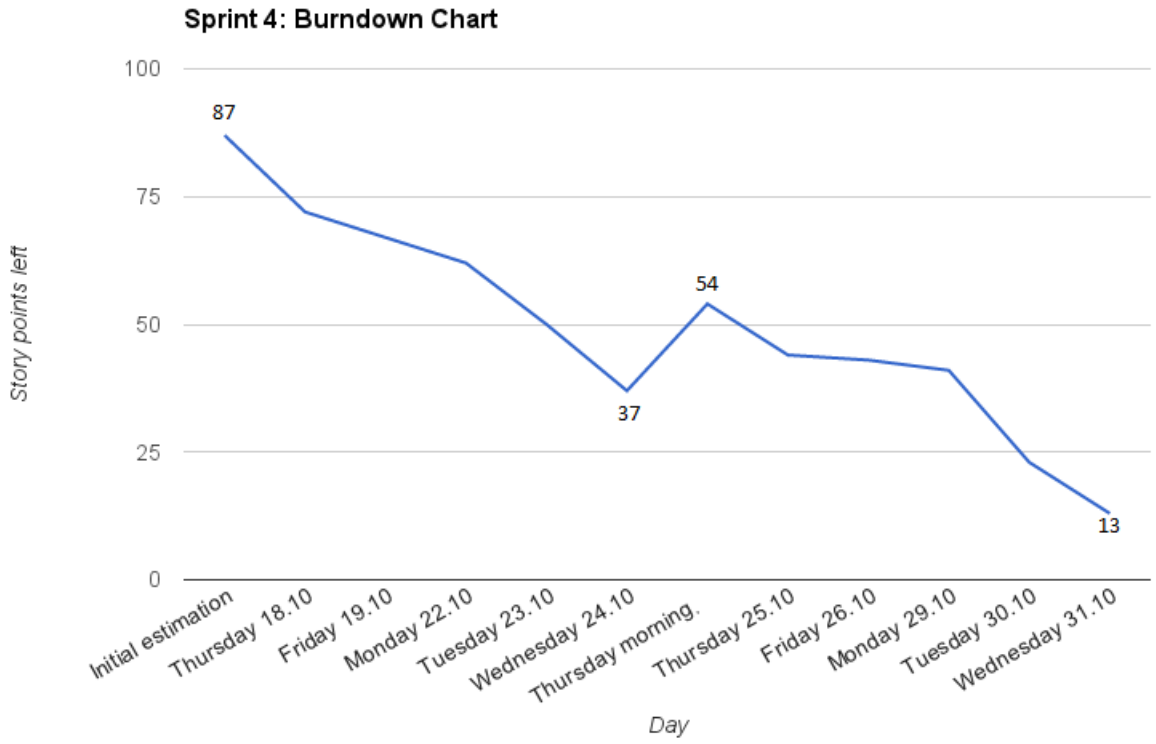


Figure 10.3: The burndown chart for the fourth sprint

would like to ask our supervisor about. The sprint backlog with all the tasks can be found in section G.4.

10.5 Sprint Retrospective

The burndown chart is presented in Figure 10.3. In this sprint we started with an initial estimation of 87 story points. In the first week we spent a lot of time on programming, and finished early with all the programming tasks. We had expected to receive the feedback from the preliminary delivery early in the sprint, but it did not show up before later. We therefore spoke with our customer, and decided to start implementing two new functions. This is the reason for the jump in the chart at Thursday morning, where we increased the story points by 17, giving us a total of 104 story points in this sprint.

In total we used 252 hours working, where 170 were on engineering tasks. We managed to do 91 story points in these hours, giving an average of 1 hour 52 minutes for each story point. In this sprint we did not finish all our story points, with a total of 13 story points

remaining. The remaining story points are distributed between ET66, making sure that the report is written using technical language (11 points), and ET84, find references to already written text (2 points). The reason for the remaining story points is that we used a bit more time on refactoring of the code and testing of the system than expected. Also, dropbox were unavailable for three hours on Tuesday 30th of October, the day we had planned to work in LaTeX. Therefore, we could not work on the engineering task regarding technical writing in our report. The two remaining tasks will be postponed to the testing week.

We underestimated the workload on especially one of the engineering tasks, ET84, find references to already written text. This was estimated as 3 story point, where we only managed to do one of them in 17 hours.

A retrospective meeting with the group was held at the 1st of November at 10:30. At this meeting we discussed our working methods. The results of the retrospective meeting were divided in three categories, where the category containing things we should stop doing were empty. The other two categories were things we should continue doing and things we should start doing.

The start section mentioned four things which we decided to focus on in the next sprint. This were improve our test coverage, working more hours, have lunch together once a week and give each other more feedback while working with documentation. The way to improve our test coverage is to make everyone focus on making tests while coding. The persons that have less hours than required will work in the weekend or evenings to get a jour. For the feedback part, when someone has written something, someone else should read through it and give feedback.

The continue section summed up what we were happy with: working together at school, selecting only one ET to work with at a time, having creative programming meetings, coming on time, helping each other and having a positive attitude, speaking English together and continue taking small breaks.

Chapter 11

Sprint 5

This was our fifth and final sprint in this project. Mirna was the only one who had not tried to be Scrum master, so she was eager to try.

This chapter is divided into the following sections. The sprint breakdown with all the meetings and important dates. The sprint planning with our overall plan for this sprint and the user stories we were going to do. The section about testing, where we discuss how we tested the tasks and the user stories we finished. Next, the result section where we write about what we finished, with some screenshots of our system, and finally, the sprint retrospective with the hours we have used on each story point, the burndown chart and reflections.

11.1 Sprint Breakdown

The sprint started with a planning meeting at the 1st of November. The customer experienced some technical problems, so we were not able to have a meeting with him before the next day, the 2nd of November. The 1st of November we did also have a meeting with our supervisor. We planned to have daily Scrum meetings with the whole group, and to work together half of Friday, Monday, Tuesday, Wednesday and Thursday. This sprint was only going to last one week, and that was the reason that we decided to work together each day.

At Thursday the 8th of November we should end the sprint with a meeting with our customer. Here, we would give a demonstration of our product. Also, at this date, a milestone is scheduled, feature freeze. This means that all functionality should be added to our program before this date, and no new functionality will be added later.

11.2 Sprint Planning

The customer had no Internet connection the day the sprint planning meeting were scheduled, so we did not know which user stories to start working with from the beginning of

Table 11.1: The current user stories in the sprint backlog

ID	User story	Overall priority	Priority this sprint
US4	As a student, I need to document the work thoroughly, so that we can write a good report of our project.	High	High
US15	As an air traffic controller, I want the system to give some feedback if an airplane is on crash course.	Low	High
US19	As an air traffic controller, I want to be able to rotate around the airplane in follow mode.	Low	Low
US20	As an air traffic controller, I want the system to give me some audio feedback.	Low	Low

the sprint. In our product backlog, only low ranked priority user stories were left, so we decided to start working with collision detection.

At the customer meeting the next day, the customer agreed with us to focus on this user story. He also came up with some new technology that we had not looked into before, namely the system responding to a users input by its own computer voice. We all agreed that such a functionality seemed really cool, and since our customer said that this was not so difficult, we decided to add a user story regarding the system responding to user input by a computer voice.

The last programming user story we should focus on was to add the possibility to rotate around an airplane that is being followed by the camera. Of course, we were also going to work with the documentation in this sprint as well.

Table 11.1 shows all the user stories we were going to work with in our last sprint.

11.3 Testing

This section contains information about the tests we conducted in this sprint. The acceptance criteria are shown in Table 11.2.

In this sprint we used integration testing to make sure that the new functionality with the system speaking back to the user could be integrated without introducing new errors. Unit testing was used to test that the system responded as intended each time a user gave a command, as well as to test the methods regarding collision detection and the rotation around the airplane.

Table 11.2: Acceptance criteria for the fifth sprint and their evaluation

ID	**Acceptance criteria**	**User story**	**Verified Finished**
AC30	The documentation should contain an overall test plan, and citation should be included where needed.	US4	Verified by group
AC31	Some feedback is displayed if two airplanes are on a crash course.	US15	Verified by customer
AC32	If an airplane is selected, one should be able to conduct a 360 degrees rotation, both vertical and horizontal, around the airplane.	US19	Verified by customer
AC33	The system should give feedback to the user regarding situations and alerts.	US20	Verified by customer

The unit testing regarding the system speaking back has been tested each time a new command was added. This has been tested by adding the command, running the system, executing the actions that leads to the command being said, and then listen to make sure that it is the correct command.

The unit testing regarding the camera rotation in following mode have been executed by running the system, start following an airplane, and then try to rotate around it in all kind of directions. This function has also been tested with extreme cases, like trying to divide by zero, and make sure that the system does not crash while this is tried.

The unit testing regarding the airplanes on a colliding course has been done by lining up the planes so they are going to crash, and then see that the system responds by notifying the user that the airplanes indeed are going to crash.

The integration tests have been done by just playing around with the system after all the new functionality has been integrated, and make sure that it behaves and responds as intended.

11.4 Results

This section describes what we did on the programming part and the documentation part in this sprint.

The main focus on this sprint was to make our program even better, without integrating a lot of new functionality. We had finished all our high and medium prioritized user stories from our product backlog (except for the documentation user story), and had only low prioritized user stories left.

The first things we did was to remove the black threshold line as a result of the usability

Table 11.3: The system's feedback to different situations

Precondition	**Example feedback**
When two airplanes are heading towards each other on a collision course	Warning: plane one and plane two will crash in less than five miutes
When a user selects an airplane	Plane one is selected
When a user alters the heading of the selected airplane	Altering the heading of plane one to 50 degrees
When a user alters the speed of the selected airplane	Altering the speed of plane one to 600 km per hour
When an airplane has less than 1000 litres of fuel left	Warning: Fuel is getting low in plane one

tests subsection 12.1.5 conducted in the previous sprint. The new calibration was placed at the user's shoulders height. This made the command "calibrate" unnecessary, and we removed that.

We focused on three main programming user stories in this sprint. The first user story was to add collision detection to our program, and to notify the user if two airplanes were on a colliding course. Figure 11.1 shows two airplanes on a colliding course, with the systems's feedback displayed at the bottom. We solved this by doing some calculation on the placement of each of the planes for the next five minutes, and if two planes were going to be at the same position in the near future, the airplanes would collide. This was then communicated to the user by the system saying that the airplanes will crash, and in how many minutes this will happen. A text is also written saying which planes will crash and when.

The second user story was to make it possible to rotate around an airplane when it was followed by the camera. This was solved by making the camera look at the airplane, and then set the camera's position according to the movements from the user. To make it go smooth, the new position of the camera was calculated from its last position, and also taking into account the airplane's and movement between two frames.

The last programming user story was about making the program able to respond to user input with a talking voice, and with a line of text at the bottom of the screen. The feedback line is shown in Figure 11.2. The Windows speech SDK offers a framework making the implementation of computer voice pretty easy. By making a class that handles the strings of text, and sending them both to the display and to the speaker class, no redundant work is done. This class will then handle both the written and oral response the user gets from the program. The system's response to the most important user actions are shown in Table 11.3.

In the documentation part we have focused on finishing one of the tasks remaining from

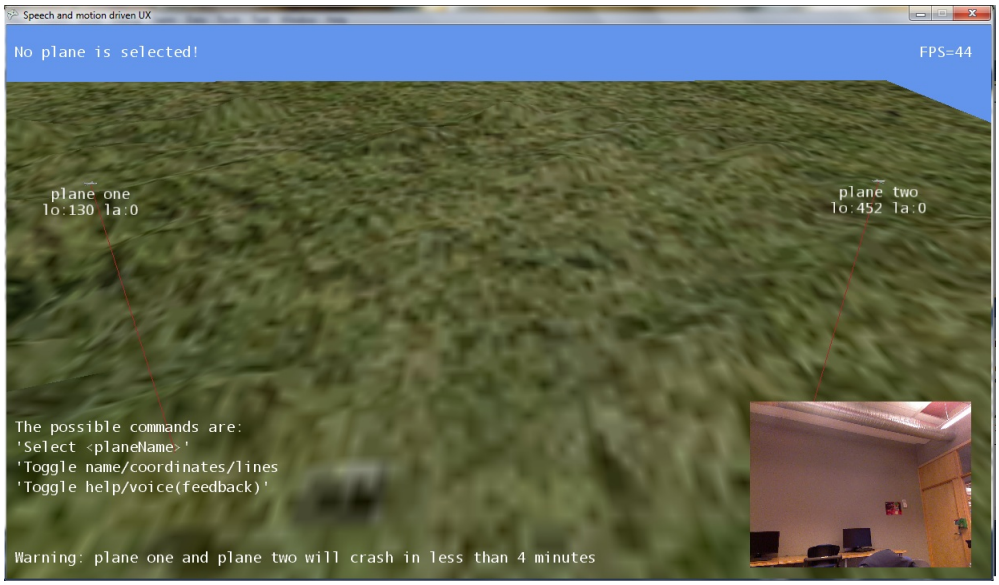


Figure 11.1: Screenshot showing the feedback given when two planes are on a colliding course

Figure 11.2: Screenshot showing the feedback given when a plane is selected

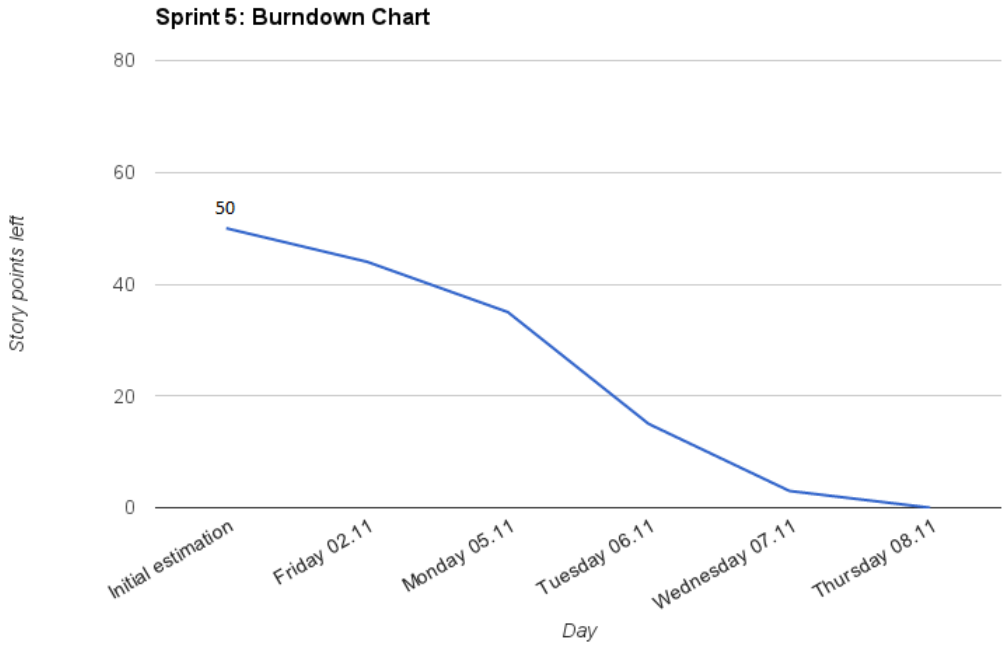


Figure 11.3: The burndown chart for the fifth sprint

sprint 4, finding references to already written text. We have also used a lot of time on changes in the documentation, as well as working more on the testing chapters, writing a chapter about the ideas we had for the system if we should develop it further and making a user manual. The sprint backlog with all the tasks can be found in section G.5.

11.5 Sprint Retrospective

The burndown chart is presented in Figure 11.3. In this sprint we started with an initial estimation of 50 story points. We managed to do all the story points during this sprint. In total we used 161 hours working, where 90 of them were used on engineering tasks. This gives an average of 1 hour 48 minutes on each story point.

Based on the working hours from previous sprints, we should have worked around 125 hours in this sprint. The reasons for the number 161, which is 36 more hours than estimated, is that we worked on Monday as well, which we have not done before.

The average time spent on each story point, 1 hour and 48 minutes, are similar to the time used in previous sprints.

We underestimated the task US104, making changes to the documentation. This was estimated as an 8, while we used 23 hours on this task, giving an average of 2 hours 53 minutes on each storypoint. Another task that was underestimated was ET84, finding

references to already written text. This was estimated as 2 story points, with a total hour use of 10. This gives 5 hours per story point. The task ET101 was overestimated. Here we estimated 5 storypoints and used 3 hours, leading to 36 minutes on each story point.

A retrospective meeting were held after the meeting with the customer. Here we discussed what we should start doing, continue doing and stop doing. We all agreed about starting with thinking about the presentation. This was the only thing mentioned under start. The continue section contained things that have been mentioned before, like continue speaking English, continue working together at school, continue being help friendly and nice to each other, and continue to arrive at time. The stop section contained one thing, we should focus more about the work being done than the hours used.

Chapter 12

Testing

In this chapter we describe the actual testing done. We describe the different testing methods used in more detail than in the overall testing plan chapter (chapter 6). Associated with these we will describe the approach, execution and the result. At the end of the chapter we present a summary result of all the tests executed.

12.1 Testing of our System

As we were using Scrum, the process of testing was a bit different than with e.g. waterfall. Having the sprints in Scrum we were to make a "finished" system at the end of each sprint, showing the customer some tangible product. We also had some user stories that were based on the product backlog, while some of the functional requirements were based on the product backlog, making the unit tests (automated tests) and integration tests (manual tests) we performed during a sprint a part of the testing of requirements. We also had some acceptance criterias for the user stories in each sprint. These were "tested" at the end of the sprint by showing the customer the result and then determine with them if the criterias were met.

As the unit and integration testing was performed during development our focus at the end when testing was system, performance, usability and acceptance testing. The system tests were based on our functional requirements for the system.

12.1.1 Unit Tests

From our discussion about different programming techniques and testing methods in section 3.3 we came to the conclusion to use the test driven development (TDD) approach when programming. In TDD one creates tests for each of the methods and constructors. The creation of tests should be done before the methods are created, and all the tests should be ran before code changes are committed to the repository. For this, we used the integrated unit test method in Visual Studio. Since there should be many tests for each

function and limited time, we have not documented these tests in the report, only in our code.

12.1.2 Integration Tests

Integration tests should be conducted when a new part of the system is added, to make sure that it functions as intended. We did some manual integration tests, after we had finished two of our major tasks, adding motion and voice control. Since we had motion and voice, it was difficult to see which signals were sent, without using a method to simulate the Kinect or the microphone. We did not have time to do this, so the tests were executed as gray box testssection 3.3.1, where we observed that the system functioned as intended.

As the integration tests were also conducted during programming and time was limited we did not documented these tests in the documentation, but their execution is mentioned in the sprint chapters.

12.1.3 System Tests

System tests are to be conducted to test the functionality of our system. The goal is to verify that the system delivers the functionality the customer wants.

Preparation and Execution

In system testing the team tested all the functional requirements of the project. The tests ensured that the system did what was expected of it, and that it could be used without errors.

The system test was conducted as one of the last tests before the acceptance test. The tests were written during the requirement specification, but they were not performed until the end. In that way we could test at the end that our requirements from the start were met in the final system. For details on our tests and the execution see Appendix I.

System test cases were made, describing exactly how the tests should be carried out. The tests were executed by a group member that had the test case as his/her responsibility. The responsible for the test case needed to make sure that the pre-condition was met before starting the test. After completing the execution part the he/she was to make sure that expected result was reached according to the pass-criteria, if not the test did not pass.

Results

All the system tests were conducted on 12th of november. All tests reached the expected output and did therefor pass with flying colours.

12.1.4 Performance Tests

Performance tests were conducted to ensure that the system provided good enough performance to what the customer wanted, that the system met the performance criterias and that it performed in a manner that was to be expected of such a system.

Preparation and Execution

We tested the performance by conducting stress and load tests where we measured the response time, uptime and frames per second.

The requirements and specifications for the systems performance were decided in the requirement gathering phase of our project, but the tests were first written later (based on the requirements). The tests were executed by a group member responsible for the performance testing. The test responsible was acquired to make sure the pre conditions were met before the test was executed and make sure that the test was executed as planned.

The tests were executed at the same period as the system tests. The different test cases for the performance tests can be found in Appendix I.

Results

All performance tests passed.

12.1.5 Usability Testing

Usability tests are performed to check if the system and its user interface is understandable for users. These tests are carried out throughout the project, where you get feedback from the customer and make corresponding changes if necessary.

Preparation and Execution

When performing the usability tests we followed the 10 point guidelines[10]. The guidelines explain how to conduct a usability test, and can be found in section J.1.

We conducted two usability tests, one early in the programming phase and one later on when we approached a complete system. Both tests were Hi-Fi (High fidelity) tests, meaning that our tests were complex prototypes of the system near the final product, as making a paper prototype for our system with few details was rather hard, given the technology needed and requirements. The first test was conducted by an external user that was not familiar with the project or the system, but had some programming experience and an over average technical understanding. The main purpose of the test was to test the usability of the motion control in our system. This was done first because the voice control-part was not yet finished.

Two persons from our group were responsible for conducting the test. One test-leader who introduced the test and one responsible for observing and taking notes. The tasks

given to the user is presented in section J.2. Before representing the task the user was also given a manual of the system. This was necessary as our system will not be intuitive and a "learn by doing" system, but rather a system that is easy to use and has a steep learning curve after reading through the manual.

After the tests were finished we asked the user to fill out a SUS(system usability scale) sheet. The sheet is presented in Figure J.4. It is a simple ten-item scale that gives a global view of subjective assessments of usability. Then we used the numbers from the SUS sheet to calculate the score. This score gives us a composite measure of the overall usability of the system. The SUS scores have a range of 0 to 100, where 100 corresponds to a very good usability[24].

Results

Below are the results from the usability test represented:

- There were some problems with the calibration of the line. The user did not know which height was to prefer for the line as she had never used the system before or gotten any information of what was best.
- One of the users had a problem with making the Kinect register the speech commands, due to a lack of American accent. This was however improved as the user got further in the task and got more used to talk in a more American accent.
- One of the users missed a form of feedback so that she could know that the command had been registered. Especially when making the alter heading and pitch command. Both users felt that it was not enough to see that the plane changed the direction or pitch, as the changes could be too small to register with the naked eye.
- Both users felt that there was difficult to see what the information on the left side was for. They did also not understand that this information was helpful to them.
- One of the users tried to deselect before selecting an airplane. She did this because she thought it had to be done before selecting another airplane. She thought this mainly because the "deselect" command was above the "select" command in the information displayed on the screen.
- Both users also felt that it was a bit difficult to know how you perform the different motion commands, having a demonstration from a person who knows the system in addition to the manual was something they felt could be a good idea.

The SUS score from our users after the usability tests were 80 and 65. The reason for the one slightly lower score is the statements provided in the SUS sheet. Statement 1 "I think I would like to use this system frequently" as the users tested are not air traffic controllers they have no need or desire to use the system frequently. As our system requires a walk-through in addition to manuals before the user uses the system for the first time there were also some inequalities regarding statement 10, that says "I needed to learn a lot of things before I could get going with the system". The user is required to read the

Figure 12.1: Screenshot showing the program with all information off

manual before use. But the user was informed of this by the testing responsible, before filling out the SUS-form, making the statement more suitable for our system. Despite these things mentioned, the score was very good, and if the statements not fitting our system that good are eliminated both scores increases significantly.

Changes

The main changes we made after the usability tests were on the presentation of the information on the screen, the feedback presented for the user, and we also made a more comprehensive manual describing the system and the use of it in more detail.

We choose to have a command line for the feedback from the system, so the user can see that its command has been registered. We also added the ability for the system to provide speech feedback. This was something the user could select/deselect. The presentation of the information on the screen was also improved by adding some ”spaces” between the different categories, and more space on the screen was saved by giving the user the possibility to remove the presentation of possible voice commands one can give the system. The presentation of different voice commands could be removed from the screen after giving a command ”toggle commands”.

We also decided to remove the black line, indicating how high the hand should be placed for the Kinect to recognize the motion. We removed the line and made it as default that

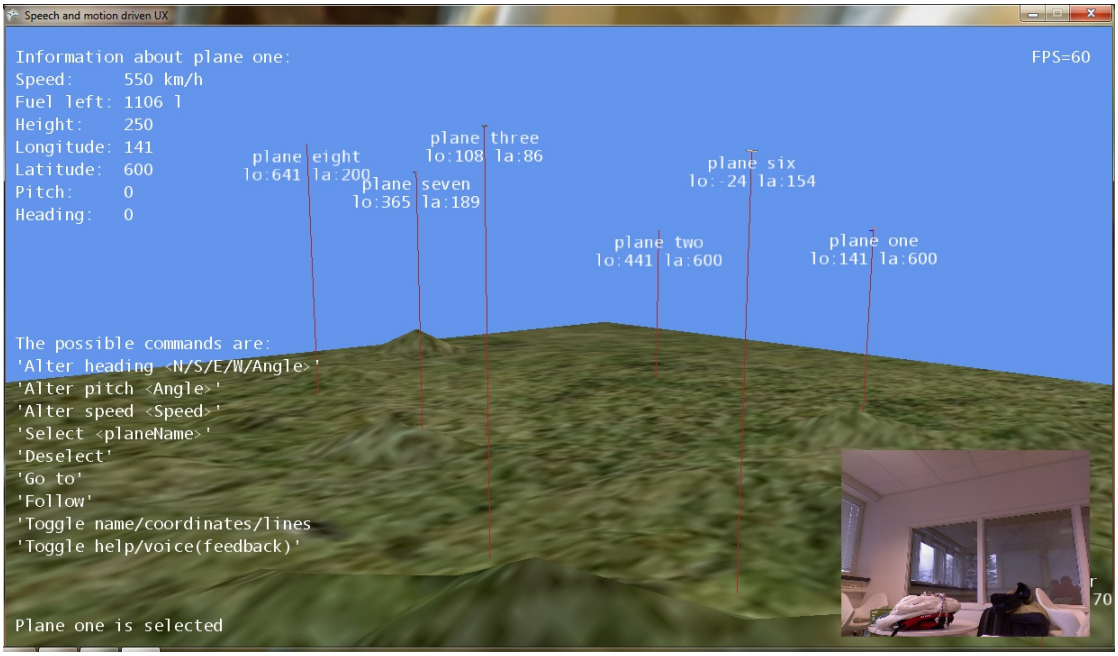


Figure 12.2: Screenshot showing the program with all information on

all movements above should-height was detected. We did this because the users struggled with the calibration of the line, as they did not know what was the best height. We also did it to make the interface better, by removing the line we removed the need for the user to have to watch the screen continuously, to make sure the hand is above the line.

In Figure 12.1 we show the system with no displaying of information and in Figure 12.2 the system with all possible information available on the screen.

12.1.6 Acceptance Test

The purpose of acceptance testing is to make sure that the customer accepts the system we have created.

Preparation and Execution

There are a number of ways to perform acceptance testing. Since we are using Scrum we have acceptance criterias that need to be fulfilled at the end of each sprint. Here we present the system part finished during that sprint, along with associated documentation.

At the end we had user acceptance testing and final acceptance test with the customer on the whole solution, where they either approve or disapprove our solution. The test was performed based on the system's requirements and the test conducted by a group member

while one representative from the customer was present through Skype.

Result

We conducted the acceptance test on the final meeting with the customer on November the 8th. The customer found no errors or deficiencies, and accepted the system based on the functional requirements.

12.2 Test Results Summary

As all of our tests passed without any problems, there was not much that needed to be changed in the code. The most changes were made with regards to the usability tests, where we changed the presentation (UI) and the user manual. To get the best results we could have had one more usability test, to check if our changes made the users more satisfied. But due to the fact that the users were not that disappointed in the UI and had a good understanding of it, and time limitations at the end of the project, we chose not to conduct one more iteration with usability testing.

Below we present a requirement traceability matrix. The table shows that all functional and some non-functional(those possible to test) requirements have been met, through testing.

The non-functional requirements that can not be verified through testing are not presented in the table. The requirements in mind are non-functional requirements 1 to 4.

Table 12.1: Requirement Traceability Matrix

	Requirement ID											
Test cases	**FR1**	**FR2**	**FR3**	**FR4**	**FR5**	**FR6**	**FR7**	**FR8**	**NR5**	**NR6**	**NR7**	**NR8**
S01	x	x		x								
S02	x	x		x								
S03	x	x		x								
S04	x		x	x		x						
S05	x		x	x								
S06	x		x	x								
S07	x		x	x	x	x		x				
S08	x		x	x								
S09	x			x			x					
P01				x					x			
P02				x								
P03				x							x	
P04	x			x								x
U01	x			x						x		

Chapter 13

Further Work

This chapter contains the visions for future development of the system and ideas for using the technology. We managed to do everything we had decided in the product backlog, but this is ideas that would have been cool to add if the system will be developed further. We have also included a section describing other ways to use the voice and motion detection technology in other scenarios.

13.1 Extending Our System

This section contains ideas for how we would extend our current system.

13.1.1 A Real Air Traffic Controllment System

In a real air traffic control system, the system should be able to handle many simultaneous users, running the system on different machines at the same time. This would require an interface towards a network, a server responsible for running the core elements of the system, some way to update each of the user's models when another user makes a change, etc.

Another issue with a real system, is the fact that controlling an air space has the risk that even a small error can result in the loss of people's lives. Therefore it is very important to assure that all the signals and actions are transmitted without any errors. There should be several error checks, both in the transmitted signals, in the code and in the actions the user inputs. If a user for instance directs two aircrafts in a direction facing towards each other, that could result in a collision, he should receive a warning, or maybe even not be allowed to perform such an action.

The system should also provide some security on the level that no unauthorized users gets access to the information, and even worse, is able to change it. This should be ensured both on the computer where the system is running, and also, he should not be able to decode the signals sent between computers, servers and planes.

The last issue with the real system would be availability. The system should have zero downtime, because downtime would in the worst possible case lead to planes crashing and people getting killed.

Architecture for the Real System

A possible architecture for the real system would have been a client-server architecture with high focus on availability, security and performance. A description of the client-server architecture can be found in subsection 5.3.3. The reason for the client-server architecture is that the system should be run simultaneously on multiple devices. It is therefore smart to have a server in the middle handling all the information from the planes and the users, and sending this to the clients using the system, and to the planes that are affected by changes. The security issue could be solved using checksums to see if the messages are received without errors. The messages sent over the network should be encrypted. Also, the data should be duplicated on different servers, to prevent a single point of failure.

The availability issue could be solved by having backup systems in case something happens to the main system. It should also have a backup power supply if the building loses its power.

Performance should be ensured both in the software and the hardware. 3D models requires a lot from the computer running it, and therefore, the hardware should be good. Also, the network links between the clients and the servers should have a high bit rate.

13.1.2 A Compass and an Altitude Metre

A compass could be added to the model. The compass must be in 3D, like the model, and always point towards north. This would make it a lot easier for the user to know which way he is looking. Also an altitude metre could be included, displaying the current height of the user's viewpoint.

To make such a compass would be like making another model, and then placing it on top of our current 3D model. The altitude metre could be a line showing a mark at the current height.

13.1.3 Better Motion Interpretation

In subsection 3.5.5 we described the possible ways of recognize motion with Kinect. Our program is based on motions above a specified line (shoulder height). To make our program more usable, we could have made the Kinect recognize gestures everywhere, but to manage that we needed to find some way to recognize when a gesture starts or stops.

13.1.4 Adding Other Aircrafts

Models of other aircrafts than airplanes could be added. This include helicopters, different types of airplanes (like Boeings, small propel airplanes etc) and military aircrafts. They should each have their own image and their own type of behaviour (different speeds, range landing areas etc).

13.1.5 Runways Where Airplanes Land and Take Off

There could be added runways where the planes can land and take off. The landing and takeoff phase should be done in a realistic way, like decreasing the speed of an airplane approaching the runways for landing. Also, no airplanes should be able to land at the same runway at the same time. An airplane can get permission to land via a speech command.

13.1.6 Weather

The model could get weather data from the actual conditions at the airport, displaying this both with a textbox or icon, as well as the model changing color to white when there is snow on the ground. Both temperature, the direction and speed of the wind and the weather condition could be displayed.

13.2 Other Usages for the technology

Here we look at other areas where the technology used in our system could be applied to create new or improve existing products.

13.2.1 Aid for Persons With Disabilities

This technology can also be used by persons with disabilities. If they for some reason are prevented from using a keyboard and a mouse, maybe because they lack hands, or even arms, they can still use the motion recognition technology. We have used a person's hands as the tracking point for the Kinect, but we could as well used feets, elbows, cheast, head or something else.

The voice recognition technology makes it possible for persons that are not able to move at all, to use a computer program. Voice commands can be the only user interaction with the program.

13.2.2 Make Dangerous Jobs More Secure

During a rock concert, a lot of persons works up under the roof of the stage, controlling the spotlights directed towards the artist. This job is dangerous in many ways: the person sits high above the stage, so if he falls, he might get killed. The sound is loud, so he might

suffer for hearing injuries after some time. Also, the temperature is high. This scenario can be solved using our system, by saying that the 3D model is the stage containing all the persons there, and the spotlight is the viewpoint. Then, commands like "follow Bruce" will make the spotlight follow Bruce Springsteen as he moves around at the stage.

This is just an example. Other dangerous working areas where this might be applied can be work subsea, in space or in hash weather conditions.

Chapter 14

Evaluation

In this chapter we evaluate the different aspects of the course and the project from the internal work process: how we have worked as a team, what we have done well, what we would change if we could do it all over again, the project goal and what we have learned.

14.1 Internal Process

During this project the group has learned more about what it means to be working in a group together as a team, with different people. We have learned more about ourselves and what type of roles we adapt to and how we work best. We have also observed strengths and weaknesses that will be good to bring in to future projects.

14.1.1 Scrum

As mentioned the group chose to use Scrum as the development methodology and got good experience in using it. It was specially good to use Scrum since our customer was in Oslo. That way the customer got updates on the system, the project and got to see the finished part made during that sprint. Our task could also easily be divided into different modules, like the 3D model, speech recognition and motion recognition. We could then focus on one module per sprint, and had a working demo for the customer meeting, held using Skype and Teamviewer.

We also experienced some minor modifications to the requirements during our development. These were easy to make as we used the Scrum methodology. By looking at the burndown chart, we felt that Scrum gave a nice overview of what everyone was working with, in addition to the morning. It was also nice to have the retrospective meetings, as this was a nice place to discuss things we felt needed improvement in the group or things we felt needed to be started/stopped.

There were however a weaknesses with using Scrum in this project: not all of our task were suited to be presented as engineering tasks, especially tasks regarding project

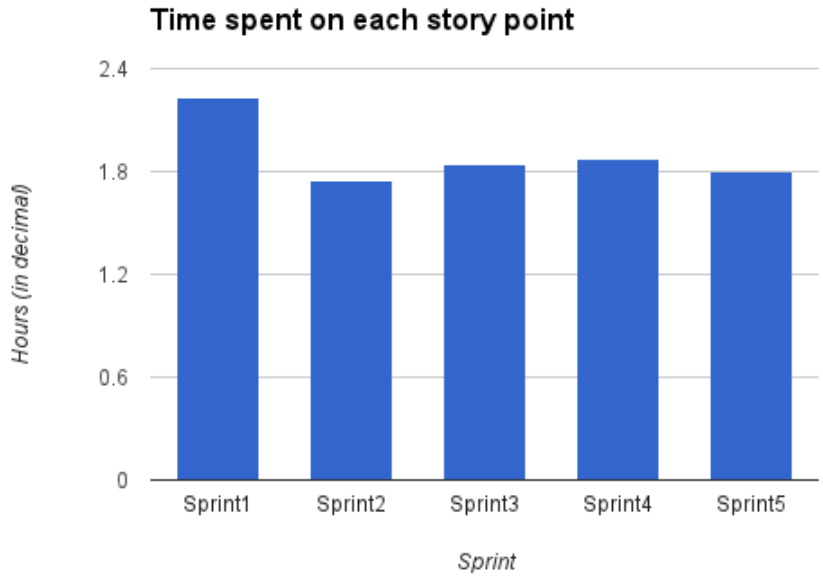


Figure 14.1: Time spent on each story point in the different sprints

management, meetings, etc. This lead to that approximately only half of our time during a sprint was spent on engineering tasks.

Figure 14.1 shows the time we spent on each story point in each sprint. Except from the in first sprint, we spent around 1 hour and 50 minutes (1.8 hours) on each story point. We were good at estimating the workload in each sprint, as the figure shows.

14.1.2 Task and Development

In the beginning of the project the task felt very big and overwhelming. But it was very interesting and we were eager to get started. Because the task was so interesting and we saw much potential in it, we were maybe not that good at narrowing and limiting the scope in the beginning. As the project also focuses much on documentation it lead to much work on both documentation and programming. We learned a lot by writing the documentation and we have improved a lot in writing technical reports, but due to the time limitation and the task we were given, we would have liked that there was a bit more focus on the development part.

Other than this we experienced that pair programming was good for us. The group members did not have prior experience in working with Kinect, XNA, 3D modelling, speech recognition, speech feedback, motion gesture detection, and only one member had experience with C#. So there was a lot to learn and understand before one could start the actual programming.

Based on the time and amount of work needed to get familiarized with new tools we are very pleased with our finished product, our effort and the task from the customer.

In the beginning of the project we made a risk analysis, where we identified all the risks and found strategies for how we would manage them if some occurred. Making the analysis was useful later in the project as we experienced some of the risks, like illness, requirements change and tools fail, when Dropbox was down. These were handled as defined in the strategies and did therefor not lead to any problems.

14.1.3 What Went Especially Well

The group dynamic was very good, and everyone got along well. We had a good communication, making it easy to ask other group members for help and discuss ambiguities. Also, no time was wasted on quarreling, thanks to the good communication.

All team members had the same goals: to get a good grade and do their best, which lead to everyone working hard. Each group member worked around 25 hours a week, as recommended by the course staff. This lead to that we managed to finish both the system and the report in the given time, without working overtime towards the end of the project. As shown in Appendix L, in total we have spent 1607 hours on this project. This gives an average of 24.7 hours per person per week.

We were also good at planning and making realistic goals based on our experience and knowledge level. Having three days a week where we worked together was also smart, that way we worked more as a group together, cooperation and discussing.

We are glad we had usability test for the system, and we got valuable feedback from the two test user. This feedback was used to improve the system in a way both the group and the customer liked.

The system was finished based on the requirements, and was approved by the customer. We managed to keep the code clean and easy to read. It was good that we wrote down and agreed on coding conventions early in the project.

14.1.4 What Could Have Been Done Better

We had a social evening in the middle of the project. This evening could have been planned earlier, so we could get to know each other better from the start. We also have some minor improvements that we see in the retrospective could have been done, we could for example have had more time set aside for everyone to learn more about all the parts of the program (like making the 3D models in Blender and the voice recognition). But as a result of limited time and a lot documentation work not everyone got time to work on this.

Thanks to the retrospective meetings after each sprint we manage to talk about improvements that could be done for the next sprint, and what had been good and should continue.

14.2 The Customer

We have had positive experience with the customer and we are very pleased with them. They have provided us with good and helpful feedback, their commitment has been good, they have been helpful throughout the project and very easy to talk to.

14.3 The Supervisor

Having an supervisor was very nice. We have not had a project like this on NTNU before, and writing a technical report at this big scope is something none us has done earlier, so having a supervisor that could help us with the documentation during the project and guide us have been important. In the start we felt there was too much feedback only on the structure of the documentation, but after explaining that we wished more concrete feedback the supervisor gave us that. The communication has been good and we were pleased with our supervisor.

14.4 Suggestions for Improvement

The workload and expectations in this course are very high, a lot of work and time is put into the project, making other courses the group members have down prioritised. For our group we managed to balance this. We experienced several times that documentation we wrote and structure of report based on suggestions in the course compendium was not to be preferred by the supervisor. Also the timetable set up in the compendium was something our supervisor did not like. We also think that having a "start up" week in the beginning of the course can be a good idea. During that week students taking the course could maybe answer an evaluation form, answering how much effort they plan on putting in the subject etc. and based on their answers here this could be used to make groups.

In our group we had an international master student, for her it was difficult and very drastic to start her first semester in a new country with this kind of group project. It was especially difficult to adapt to so many new things, regarding both academical, technical and cultural. A suggestion can be maybe to move this project from the first to the last semester for these students.

The group agrees upon that having a project like this is very educational and interesting, and that it prepares us for situations we can meet in the "real world."

14.5 Conclusions About the Product

We felt that we solved the task in a good way, proving the concept that both motion and voice can be integrated with a 3D model in a satisfying manner. We managed to develop a system where motion and voice where used to control the 3D model we created.

The 3D model in our finished product consisted of a number of airplanes and the environment where they operated (a ground with mountains and a sky). Both motion and voice were used for controlling the 3D model. Motion controlled the entire model, while speech was mostly used to control a single airplane.

For both motion and voice recognition, the implementation were realtively straight forward. The reason was that there exists good SDKs for this, abstracting away the most difficult parts, and providing an intuitive interface towards the programmer.

Appendix A

System Requirements

Below you will find the system requirements [83]. First a list with the minimum requirements for running the application, and then a list of required software for further development of the system.

A.1 System Requirements to Run the Program

- Windows 7
- Graphics card that supports DirectX 9.0c and Shader Model 1.1[82]
- 32-bit (x86) or 64-bit (x64) processor
- Dual-core 2.66-GHz or faster processor
- Dedicated USB 2.0 bus
- 2 GB RAM
- A Microsoft Kinect for Windows sensor

A.2 Software Requirements to Continue Developing on the Project

- Microsoft Visual Studio 2010 Express or other Visual Studio 2010 edition
- .NET Framework 4.0
- Microsoft Speech Platform SDK v11
- Microsoft XNA Framework

Appendix B

Templates

This chapter contains the templates we used for meeting agendas, both with our supervisor and the customer. It does also contian the template we used for the weekly status report we sent to our supervisor each week.

B.1 Agenda Supervisor Meeting

The report contained these elements:

1. Meeting information
   - Date
   - Time
   - Room
   - The attendees: Supervisor, group
   - The name of the person taking minutes
2. The agenda:
   (a) Approval of agenda
   (b) Approval if minutes from last supervisor meeting
   (c) Weekly status report
   (d) ...
   (e) ...
   (f) Plans for next week
   (g) Close meeting

B.2 Agenda Customer Meeting

The report contained these elements:

1. Meeting information
    - Date
    - Time
    - Room
    - The attendees: Customer, group
    - The name of the person taking minutes
2. The agenda:
    (a) Approval of agenda
    (b) Demonstration of product
    (c) Review of previous sprint
    (d) The next sprint with sprint backlog and acceptance criteria
    (e) ...
    (f) ...
    (g) Next meeting
    (h) Close meeting

B.3 Weekly Status Report

The weekley status report contained this elements:

1. Report information
    - Week number
    - Project name
    - Customer
    - The group number
    - Name of group members
2. Status information
    (a) Summary
    (b) Work done this period
        i. Milestones reached this week
        ii. Status of documents
        iii. Meetings
    (c) Problems and issues
    (d) Planning for the following period
        i. Work
        ii. Meetings
3. A table containing the hours worked previous week, presented in a table with the hours planned to work that week

Appendix C

Risk Assessment

This appendix consists of a detailed description of the different risks in the project, presented in tables.

Explanation of the Different Fields in the Tables:

Risk ID: Gives the risk an ID based on if it is an external (E) or internal (I) risk.
Activity: Which part of the project will be affected by the risk.
Risk factor: Information about the problem that causes the risk
Consequences: The consequence of the problem, how the effect will be on the group and project. The concequence field starts with a degree: L, M or H. The letters represent low, medium and high and are measures for the probability of the problem/risk to occure.

L: May cause some increase in the workload.
M: May cause increase in workload and deadlines from not being reached.
H: The project may suffer greatly and work may halt.

Probability: Gives a measure of how how probable it is that the problem occures, divided in three degrees as described above.
Strategy and actions: What we will do in our project to prohibit the problem of happening and what we will do if it happens.
Deadlines: When the problem may be a risk.
Responsible: The person responsible for taking necessary measures and actions that is needed in the situation.

C.1 Internal

This section contains all internal risks. They are shown in Table C.1 to Table C.10.

Table C.1: Illness

Risk ID	I1
Activity	All
Risk factor	Illness in the team
Consequences	M: May cause more workload on other members or postponement of the deadline
Probability	M
Strategy	The workload will be distributed among others in the team, and the ill person should frequently be given updates on the project. Making his comeback easier for him/herself and the team.
Deadline	Continously
Responsible	Secretary and PM

Table C.2: Long-term leave

Risk ID	I2
Activity	All
Risk factor	Internal/external person has a long-term leave due to illness, vacation-trip or other causes.
Consequences	H: If an external person has a long-term leave causing the project to continue without a customer or supervisor it can have a dramatic effect. The specifications may change dramatically, causing the project to differ greatly from the customers wantings.
Probability	L
Strategy	Organize a meeting and make changes to the team schedule accordingly.
Deadline	Continously
Responsible	PM

Table C.3: Unable to reach deadline

Risk ID	I3
Activity	Deadlines
Risk factor	Group not able to reach a set deadline.
Consequences	H: The grade may get negatively affected or the customer will lose trust in the team.
Probability	M
Strategy	Have clear deadlines that are realistic, and always have a buffer-time before every deadline
Deadline	Continously
Responsible	PM and Scrum master

Table C.4: Overworked team members

Risk ID	I4
Activity	All
Risk factor	Overworked team members because too much workload was put on them, or other reasons
Consequences	M: May cause interference in the team and inhibit the team work making the progress slower.
Probability	M
Strategy	Make sure that no one is excluded or overrun by other team-members. Maintaining a good communication flow is therefore important.
Deadline	Continously
Responsible	PM

Table C.5: Members not able to complete task

Risk ID	I5
Activity	All
Risk factor	Team members not able to complete their task
Consequences	H: Team member not completing their task, causing others to have to take over their task and in that way increasing the workload.
Probability	M
Strategy	Have good communication within the group, and in that way detect if someone has problems finishing their task. We use scrum and have meetings every morning, and in that way keeping a overview of what other team members are doing.
Deadline	Continously
Responsible	PM

Table C.6: Lack of knowledge

Risk ID	I6
Activity	All
Risk factor	The lack of knowledge needed for completing a task.
Consequences	H: The work may halt, not be done and time gets wasted.
Probability	M
Strategy	Every group member spends the time needed on self-study to earn the new tools and methods needed to do the project.
Deadline	Continously
Responsible	PM and every group member

Table C.7: Lack of HW

Risk ID	I7
Activity	All
Risk factor	Lack of needed HW
Consequences	M: Software can not run on any hardware
Probability	M
Strategy	Use hardware available from NTNU or share using other group members hardware.
Deadline	Continously
Responsible	PM and research manager

Table C.8: Members busy with other subjects

Risk ID	I8
Activity	All
Risk factor	Members are busy with other subjects
Consequences	M: Team members too busy with other subject causing them not to have much time for the project, leading to unfinished tasks and/or work halting.
Probability	M
Strategy	Have good communication within the group, detecting if someone has too much to do.
Deadline	Continously
Responsible	PM

Table C.9: Room reservation

Risk ID	I9
Activity	All
Risk factor	Reservation of room for meetings with group and with customer
Consequences	M: Work may be delayed if we don't get a room every time we wish to have a meeting ot schedule a work day.
Probability	M
Strategy	Try to book rooms further in time as early as possible. Or sit in the P15 building if no rooms are available.
Deadline	Continously (before every meeting)
Responsible	PM

Table C.10: Internal conflict

Risk ID	I10
Activity	All
Risk factor	Internal conflict arises
Consequences	H: The work done by the member may be of bad quality, or done incorrectly. The customer and supervisor may think less of us.
Probability	M
Strategy	Maintain a good communication within the group. We use scrum and have meetings every morning, and in that way keeping a overview of what other team members are doing.
Deadline	Continously
Responsible	PM and Scrum master

C.2 External

This section contains all external risks. They are shown in Table C.11 to Table C.15.

Table C.11: External conflict

Risk ID	E1
Activity	All
Risk factor	Conflicts arise
Consequences	M: May cause interference between the team and customer or others, and inhibit the teamwork making the progress slower.
Probability	L
Strategy	Contact supervisor at IDI for advice.
Deadline	Continously
Responsible	PM

Table C.12: Changes in requirements

Risk ID	E2
Activity	All
Risk factor	The customer changes the requirements or conditions.
Consequences	H: The workload increases.
Probability	M
Strategy	Be clear from the start on what the requirements are
Deadline	Continously
Responsible	Customer contact

Table C.13: Lack of input

Risk ID	E3
Activity	All
Risk factor	Lack of input
Consequences	H: If the team doesn't get enough input from the customer at a early phase misunderstandings can arise and the project may head towards a wrong road.
Probability	L
Strategy	Have meetings with customer often and contact the customer if the team experience ambiguities related to the requirements
Deadline	Continously
Responsible	Customer contact

Table C.14: Tools fail

Risk ID	E4
Activity	All
Risk factor	Tools fail
Consequences	M: Time may get lost and work not done in time.
Probability	M
Strategy	There will be done research early in the project, and in that way the best tools for the group based on the result from the researches will be used.
Deadline	Continously
Responsible	Research manager

Table C.15: Loss of data

Risk ID	E5
Activity	All
Risk factor	Loss of data
Consequences	H: If data gets lost many hours of work vanishes with it. Without any restring the workload may increase dramatically for all the members of the team.
Probability	L
Strategy	Take backups of everything.
Deadline	Continously
Responsible	Document manager

Appendix D

Acronyms

Here are the acronyms we have used in this report:

- 2D - Two Dimensions
- 3D - Three Dimensions
- AC - Acceptance Criteria
- DLL - Dynamic-Link Library
- ET - Engineering Task
- GUI - Graphical User Interface
- HW - Hardware
- IEEE - Institute of Electrical and Electronics Engineers
- IDE - Integrated Development Environment
- IDI - Department of Computer and Information Science
- IT - Information Technology
- MVC - Model-View-Controller
- NTNU - Norwegian University of Science and Technology
- OS - Operating System
- RGB - Red, Green and Blue
- SDK - Software Development Kit
- SVN - Subversion
- SW - Software
- TFS - Team Foundation Server
- TTS - Text-To-Speech
- UI - User Interface
- US - User Story
- UX - User Experience
- WPF - Windows Presentation Foundation
- XNA - XNA Not Acronym (originally Xbox New Architecture [39])

Appendix E

Coding Conventions

Language

Comments, classes, methods, variables and test should all be written in English.

Layout Conventions

One statement per line. Similar short variables can be declared on the same line to save space, but variables should not be defined on the same line. Curly brackets for starting and ending functions and if statements should come on its own line.

```
// Declaring multiple variables could be done on the same line
int NumberA, NumberB, NumberC;

// Definding multiple variables should be done on different lines

int NumberD = 5;
int NumberE = 10;
int NumberF = 7;
```

Naming Conventions

Classes should have camel-case names, starting with uppercase:

```
public class ExampleClass()
```

Methods should have camel-case names, starting with uppercase:

```
public void ExampleMethod()
```

Global variables should have camel-case names, starting with uppercase:

```
public int ExampleVariable
```

Local variables, argumens and other variable with small scopes should have camel-case names, starting with lowercase and be short.

```
public int ExampleMethod(int exampleArg)
{
        int exampleLocalVar = exampleArg + 1;
        return exampleLocalVar;
}
```

Constants should have all uppercase names, using underscore to separate each word:

```
const int EXAMPLE_CONSTANT = 1;
```

Variable Scopes

The scopes of variables should not be bigger than needed.

```
 //Do this
public int CountTo(int number)
{
        int counter = 0;
        while( counter <= number)
        {
                Console.WriteLine(counter);
                counter ++;
        }

}

//Not this (Counter is only used inside CountTo)
int Counter = 0;
public int CountTo(int number)
{
        Counter = 0;
        while( counter <= number)
        {
                        Console.WriteLine(counter);
                        Counter ++;
        }

}
```

Expressions

There should be parentheses around every subexpression if there is more than one:

```
if( ( a == b ) && ( b < 10 ) )
```

If expression with short single line statements do not need curly brackets:

```
if( a == b )
        DoOneThing();
```

Else they need curly brackets:

```
if( a == b )
{
        DoOneThing();
        DoOneMoreThing();
        DoEvenMoreStuff();
}
```

Testing

Similar asserts can be done in the same test method, so that the test files do not become unnecessarily large. Store expected and actual variable before the assertion, to make the test more clear. Test names should be self-explanatory, and all names for testing methods should end with Test.

```
[TestMethod()]
public void AddTest()
{
        int actual, expected;
        expected = 10;
        actual = ExampleClass.Add(4, 6);
        Assert.AreEqual(expected, actual);

        expected = 0;
        actual = ExampleClass.Add(4, -4);
        Assert.AreEqual(expected, actual);
}
```

Commenting

Start commenting at same indentation as normal code. Add one space before the comment in single line comments.

```
// Single line comment
```

All methods should have a comment in front of it on the formed showed under. It should have a short summary, a list of all parameters, and what is returned

```
/// <summary>
/// Get airplane by its name.
/// </summary>
/// <param name="name">Name of the airplane</param>
/// <returns>The airplane with that name</returns>
public static void SelectPlaneByName(String name)
{
        // ....Do something
        return plane;
}
```

Refactoring

Refactor the code before committing if possible. This is true for both normal code and test, the code is easier to maintain this way.

```
// This code does the same thing three times.
public int RepeatingCode(int a; int b; int c)
{
        a = Add(a, 25);
        a = Divide(a, 5);
        a = Add(a, 10);

        b = Add(b, 25);
        b = Divide(b, 5);
        b = Add(b, 10);

        c = Add(c, 25);
        c = Divide(c, 5);
        c = Add(c, 10);

        return a + b + c;
}

// This is more compact, and has the code in one place
public int RefactoredCode(int a; int b; int c)
{
        a = HelperMethod(a);
        b = HelperMethod(b);
        c = HelperMethod(c);
        return a + b +c;
}

public int HelperMethod(int n)
{
```

```
        n = Add(n, 25);
        n = Divide(n, 5);
        n = Add(n, 10);
        return n;
}
```

Committing to Repository

When committing to repository write very short what you did and which ET you work on. Commit at least one time for each finished task. Check automatic tests before committing. Like: ET6: Added rotation to airplanes and ET4: Fixed position bug in AirPlane class.

Code Inspection

Inspecting the code is part of the check before setting an user story as done. This should always be done by another person than the one who programmed it. The inspector should look for mistakes and see that the code is clean and that it dont need a refactoring.

Appendix F

Product Backlog

The product backlog is shown in Table F.1.

Table F.1: The product backlog

ID	**User story**	**Overall priority**
US1	As a programmer, I want a good and fast testing environment, so that I know that the code is correct after implementing and refactoring	High
US2	As an air traffic controller, I want to see moving airplanes on my screen in a 3D model, so that I know where the plane are in the air	High
US3	As a programmer, I need to learn the Kinect framework, so that I can implement the system to support Kinect	High
US4	As a student, I need to document the work thoroughly, so that we can write a good report from our project	High
US5	As an air traffic controller, I want to be abe to see the ground	High
US6	As an air traffic controller, I want to be able to rotate the camera, so that I can get a better overview of where the planes are relative to one another	High
US7	As an air traffic controller, I want to use hand gestures to change the camera direction and position, so that I can control the system without a keyboard	High

US8	As an air traffic controller, I want to give speech commands to select an airplane, so that I can give it more commands later	High
US9	As an air traffic controller, I want to give speech commands to an already selected airplane, so that I can alter its direction of movement	High
US10	As an air traffic controller, I want to see the names of the different airplanes, so that I can distinguish them in an easy way	Medium
US17	As an airplane, I should display a list with the airplanes data and additional commands when selected	Medium
US11	As an air traffic controller, I want to see the airplanes' coordinates, so that I get a better feeling where the airplanes are	Medium
US18	As an air traffic controller, I want to see some terrain, making it easier to see how the planes are positioned relative to each other	Low
US14	As an air traffic controller, I want to be able to move the camera close to the selected airplane with a speech command, so that I can easily move towards an airplane	Low
US15	As an air traffic controller, I want the system to give some feedback if an airplane is on crash course	Low
US19	As an air traffic controller, I want to be able to rotate around the selected airplane in follow mode	Low
US20	As an air traffic controller, I want the system to give me some audio feedback	Low

Appendix G

Detailed Sprint Backlogs

In this chapter we present all the sprint backlogs, one for each sprint. The sprint backlog are shown in Table G.1 to Table G.5.

G.1 Sprint 1

Table G.1: Backlog for sprint 1

ID	**Engineering Task**	**User Story**	**Estimated story points**	**Story points left**	**Hours used**
ET1	Improve knowledge of unit and functional testing	US1	8	0	22
ET2	Research C# frameworks for doing stubbing or mocking in C# testing	US1	3	0	3
ET3	Set up testing environment	US1	5	0	2
ET4	Create a plane class with position, direction and velocity	US2	3	0	10
ET5	Create a simple plane 3D model	US2	3	0	3
ET6	Alter the direction of the planes	US2	1	0	4
ET6A	Select planes by name	US2	3	0	13
ET7	Make the planes face the direction they are going	US2	3	0	16
ET8	Create an axis class to distinguish where the planes are	US2	3	0	3
ET10	Research mocking Kinect	US3	8	2	4

ET11	Learn to program with basic gestures	US3	13	13	0
ET12	Find the benefits of using the official SDK vs the open SDK	US3	2	0	3
ET13	Write a short paragraph about similar products in chapter 1	US4	1	0	5
ET14	Write about coding conventions in chapter 2	US4	2	0	4
ET15	Write about existing technologies in chapter 3	US4	2	0	4
ET17	Write chapter 2-4 in LaTeX	US4	5	0	20
ET18	Write about the different testing methods	US4	2	0	8
ET19	Write about extra tools used in the project	US4	3	0	8
ET20	Write about the quality assurance	US4	2	0	7
ET21	Write about framework we are going to use	US4	2	0	2
ET22	Write about Itera consulting	US4	1	0	3
ET23	Write about Git, SVN, TFS	US4	3	0	3
ET24	Write about the first sprint, up to testing	US4	3	0	8
Total			**81**	**15**	**147**

G.2 Sprint 2

Table G.2: Backlog for sprint 2

ID	Engineering Task	User Story	Estimated story points	Story points left	Hours used
ET10	Document our decision on not mocking the Kinect	US4	2	0	4
ET25	Write retrospect from sprint 1	US4	2	0	3
ET26	Write about testing in sprint 1	US4	2	0	2
ET27	Write sprint introduction, breakdown and planning for sprint 2	US4	3	0	1
ET28	Write in sprint 2 about testing and results, and the first part of the retrospective	US4	3	0	5
ET29	Write more information about the milestones (chapter 2)	US4	1	0	1
ET30	Risks (chapter 2) Write the most important ones, their description and solution	US4	3	0	3
ET31	Write about black, white and grey box testing	US4	2	0	2
ET32	Reasons for chosen architecture	US4	3	0	5
ET33	Put everything we have so far of documentation in LaTeX and structure it	US4	13	0	40
ET42	Write about testing in our project	US4	1	0	3
ET43	Write about choice of testing in prestudy	US4	1	0	1
ET44	Write down our definition about sprint priorities	US4	1	0	1
ET45	Make changes to already written documentation	US4	5	0	13
ET46	Look for errors in the whole documentation	US4	5	0	12
ET34	Create a simple flat, green ground model using Blender	US5	2	0	1

Appendix H

Level Test Plan from IEEE829

Level Test Plan Outline (full example)

1. Introduction

1.1. Document identifier

1.2. Scope

1.3. References

1.4. Level in the overall sequence

1.5. Test classes and overall test conditions

2. Details for this level of test plan

2.1 Test items and their identifiers

2.2 Test Traceability Matrix

2.3 Features to be tested

2.4 Features not to be tested

2.5 Approach

2.6 Item pass/fail criteria

2.7 Suspension criteria and resumption requirements

2.8 Test deliverables

3. Test management

3.1 Planned activities and tasks; test progression

3.2 Environment/infrastructure

3.3 Responsibilities and authority

3.4 Interfaces among the parties involved

3.5 Resources and their allocation

3.6 Training

3.7 Schedules, estimates, and costs

3.8 Risk(s) and contingency(s)

4. General

4.1 Quality assurance procedures

4.2 Metrics

4.3 Test coverage

4.4 Glossary

4.5 Document change procedures and history

Figure H.1: Overall Test Plan taken from IEE829[22]

ET35	Create a class for the ground which places it into our existing environment	US5	2	0	1
ET36	Fetch the Kinect from the postal office	US3	1	0	1
ET37	Set up an XNA environment connected to the Kinect SDK and to the Kinect	US3	8	0	1
ET38	Learn to recognise any motion		8	0	15
ET39	Learn to recognise basic hand motions	US3	5	0	12
ET40	Make a simple demo moving a box with your hand	US3	8	0	6
ET41	Research the Kinect SDK's speech recognition ability	US3	3	0	7
Total			**84**	**0**	**147**

G.3 Sprint 3

Table G.3: Backlog for sprint 3

ID	Engineering Task	User Story	Estimated story points	Story points left	Hours used
ET47	Write the abstract in the report	US4	2	0	4
ET48	Write summary from retrospective meeting from the second sprint	US4	1	0	1
ET49	Write more about Kinect in the pre study chapter, and find suitable figures	US4	3	0	5
ET50	Make a list of the sources/references in LaTeX	US4	5	0	6
ET51	Read about scientific writing, and do spell checking and proofreading of the report		13	0	25
ET52	Write about the tactics for the architectural drivers	US4	8	0	5
ET53	Make views according to the 4+1 view model, and write about them	US4	8	0	14.5
ET54	Add additional testing plans	US4	5	0	3
ET55	Make changes to existing documentation	US4	8	0	33
ET56	Add references to all the figures and tables in the LaTeXdocument	US4	3	0	6
ET62	Write sprint introduction, breakdown and planning the sprint	US4	1	0	1
ET63	Write about testing and results, and the first part of the retrospective chapter	US4	3	0	3
ET64	Write about 3D research in prestudy	US4	3	0	5.5
ET57	Create a basic camera class with position and direction of view	US6	5	0	10

ET58	Create methods for moving the camera (sideways and zooming)	US6	5	0	8
ET59	Create methods to rotate the camera	US6	3	0	8
ET60	Create methods for moving the camera (sideways and zooming) using the Kinect	US7	5	0	10
ET61	Create methods to rotate the camera using the Kinect	US7	5	0	10
Total			**86**	**0**	**158**

G.4 Sprint 4

Table G.4: Backlog for sprint 4

ID	Engineering Task	User Story	Estimated story points	Story points left	Hours used
ET65	Making changes to the documentation according to the reviews from the preliminary delivery	US4	13	0	21
ET66	Making sure that the documentaion is written using technical language	US4	13	11	8.5
ET67	Finish the abstract	US4	3	0	3
ET68	Add aditional testing plans	US4	8	0	25.5
ET69	Write summary from retrospective meeting from the third sprint	US4	1	0	1
ET70	Write sprint introduction, breakdown and planning of the sprint	US4	2	0	2.5
ET84	Find references to already written text	US4	3	2	17
ET71	Write about testing and results, and the first part of the retrospective chapter	US4	5	0	5
ET72	Integrate speech recognition into our program	US8	5	0	6
ET73	Add a variable saying if the plane is selected and change the color of the selected plane	US8	3	0	6
ET74	Make sure that only one plane is selected at the time	US8	2	0	1
ET75	Add a fuel variable to the planes which decreases at a given rate	US17	1	0	0.5
ET76	Add a text box that shows the data of the selected plane	US17	3	0	4
ET77	Add a text box that shows the available speech commands for the selected plane	US17	3	0	2.5

ET78	Add speech commands for turning	US9	3	0	14
ET79	Change so that the rotation of the airplane is done gradually when turning	US9	5	0	12
ET85	Change so that the acceleration of the airplane is done gradually	US9	2	0	4
ET80	Draw the name of the airplanes above them	US10	5	0	3
ET81	Make "Toggle name" voice command, that switches showing names on and off	US10	2	0	4
ET82	Draw the coodinates of the airplanes above them	US11	5	0	4
ET83	Make a "Toggle coordinates" voice command	US11	2	0	1
ET86	Make a method in Camera to make it jump close to the selected airplane, and look at it	US14	2	0	8
ET87	Add voice command for jumping to theselected plane	US14	1	0	1.5
ET88	Make a method in Camera to follow and unfollow a selected airplane	US14	3	0	3
ET89	Add voice command for follow and unfollow a selected airplane	US14	1	0	3
ET90	Add a texture for the surface of the grass model	US18	3	0	3
ET91	Make some height differences in the surface of the grass model	US18	5	0	6
Total			**104**	**13**	**170**

G.5 Sprint 5

Table G.5: Backlog for sprint 5

ID	Engineering Task	User Story	Estimated story points	Story points left	Hours used
ET104	Making changes to the documentation	US4	8	0	23
ET84	Find references to already written text	US4	2	0	10
ET91	Write about the retrospective meeting from the last sprint	US4	1	0	1
ET92	Write the sprint's introduction, planning and breakdown	US4	2	0	1
ET93	Write about testing and results in the sprint, and the first part of the retrospective section	US4	5	0	4
ET94	Write an overall testing plan	US4	2	0	5
ET95	Write the further work chapter	US4	2	0	3
ET102	Write about usability testing conducted	US4	2	0	3
ET103	Write a (user) manual for the system	US4	2	0	4
ET96	Make collision path detection for airplanes heading towards other airplanes	US15	8	0	13
ET97	Make collision path detection for airplanes heading into the ground	US15	3	0	2
ET98	Make it possible to rotate horizontal around the airplane in follow mode with fixed length from the airplane	US19	3	0	8
ET99	Make it possible to rotate vertical around the airplane in follow mode with fixed length from the airplane	US19	3	0	7
ET100	Import the audio engine to the program	US20	2	0	3
ET101	Make strings the system can read for all kind of situations	US20	5	0	3
Total			**50**	**0**	**90**

Appendix I

Test Cases

Table I.1: Test S01

ID	S01
Name	Change camera viewpoint.
Dependencies	
Responsible	Eirik M. Hammerstad
Type	System test
Description	We should be able to adjust the viewport of the camera to the right, by doing a pre-defined gesture.
Pre-Condition	All integration and unit-tests has passed.
Execution	1. Start the system 2. Stand in front of the camera. 3. Hold left hand above shoulder-height 4. Make a gesture with the left hand from left to right. 5. Make a gesture with the left hand from right to left.
Expected result	3 The viewpoint of the camera is moved to the right. 4 The viewpoint of the camera is moved to the left.
Actual result	3 Viewpoint moved to the right. 4 Viewpoint moved to the left
Accepted	Yes
Priority	High

Table I.2: Test S02

ID	S02
Name	Change camera zoom.
Dependencies	
Responsible	Eirik M. Hammerstad
Type	System test
Description	We should be able to adjust the zoom of the camera, by doing a pre-defined gesture.
Pre-Condition	All unit-tests and integration tests has passed.
Execution	1. The system is started. 2. Stand in front of the camera 3. Hold left hand above shoulder-height 4. Move the left hand forward to zoom in. 5. Move the left hand backwards to zoom out
Expected result	4. The area has been zoomed in on. 5. The area has been zoomed out.
Actual result	4. The area is zoomed in on 5. The area is zoomed out
Accepted	Yes
Priority	High

Table I.3: Test S03

ID	S03
Name	Change camera rotation.
Dependencies	
Responsible	Eirik M. Hammerstad
Type	System test
Description	We should be able to adjust the camera rotation by doing a predefined gesture.
Pre-Condition	All unit-tests and integration tests has passed.
Execution	1. The system is started. 2. Stand in front of the camera 3. Hold left hand above shoulder-height 4. Move the right hand away from the screen to rotate camera up. 5. Move the right hand towards the screen to rotate camera down.
Expected result	4. The viewpoint of the camera changes upwards. 5. The viewpoint of the camera changes downwards.
Actual result	4. The viewpoint of the camera moved upwards. 5. The viewpoint of the camera moved downwards.
Accepted	Yes
Priority	High

Table I.4: Test S04

ID	S04
Name	Select desired airplane.
Dependencies	
Responsible	Eirik M. Hammerstad
Type	System test
Description	The user should be able to select a desired airplane to make a command at, using a predefined speech command
Pre-Condition	
Execution	1. Start the system 2. Say "Select plane" + <Plane ID>
Expected result	The desired airplane should be marked in a red color, showing that it has been selected.
Actual result	The airplane was selected and marked in red.
Accepted	Yes
Priority	High

Table I.5: Test S05

ID	S05
Name	Change airplane heading.
Dependencies	S04
Responsible	Eirik M. Hammerstad
Type	System test
Description	We should be able to alter the direction of a specified plane by using predefined voice commands
Pre-Condition	An airplane has been selected.
Execution	1. Start the system 2. Say "Alter heading North" 3. Say "Alter heading East" 4. Say "Alter heading West" 5. Say "Alter heading South" 6. Say "Alter heading 0"
Expected result	2. The selected airplane changes its heading to North. 3. The selected airplane changes its heading to East. 4. The selected airplane changes its heading to West. 5. The selected airplane changes its heading to South. 6. The selected airplane changes its heading to 0 degrees.
Actual result	2. The selected airplane changed its heading to North. 3. The selected airplane changed its heading to East. 4. The selected airplane changed its heading to West. 5. The selected airplane changed its heading to South. 6. The selected airplane changed its heading to 0 degrees.
Accepted	Yes
Priority	High

Table I.6: Test S06

ID	S06
Name	Change airplane pitch.
Dependencies	S04
Responsible	Eirik M. Hammerstad
Type	System test
Description	We should be able to alter the direction of a specified plane by using predefined voice commands
Pre-Condition	An airplane has been selected.
Execution	1. Start the system 2. Say "Alter pitch" + <angle> (+/- 15 degrees)
Expected result	The selected airplane changes its pitch angle at the given amount.
Actual result	The airplane changed the pitch to the defined angle.
Accepted	Yes
Priority	High

Table I.7: Test S07

ID	S07
Name	Show information
Dependencies	
Responsible	Eirik M. Hammerstad
Type	System test
Description	The user should be able to see different information presented on the screen after giving some predefined voice commands.
Pre-Condition	All information presentation is turned off
Execution	1. Start the system. 2. Say "Toggle names" 3. Say "Toggle coordinates" 4. Say "Toggle lines" 5. Say "Toggle help" 6. Say "Toggle voice"
Expected result	2. Names of the different airplanes are displayed below the airplanes. 3. The coordinates of the different airplanes are displayed below the airplanes. 4. A red line pointing to the ground should be displayed. 5. The system displays possible voice-commands. 6. The system provides voice feedback.
Actual result	2. The names of the airplanes were displayed. 3. The coordinates were displayed. 4. A red line pointing to the ground was displayed. 5. The system displayed possible voice-commands. 6. The system provided voice feedback.
Accepted	Yes
Priority	Medium

Table I.8: Test S08

ID	S08
Name	Follow airplanes
Dependencies	S04
Responsible	Eirik M. Hammerstad
Type	System test
Description	The user should be able to follow a selected airplane, and to go directly to the selected airplane.
Pre-Condition	An airplane has been selected.
Execution	1. Say "Follow" 2. Select another airplane. 3. Say "Go to"
Expected result	1. Follows the airplane 3. Jumps to the airplane
Actual result	1. Followed the airplane 3. Jumped to the airplane
Accepted	Yes
Priority	Low

Table I.9: Test S09

ID	S09
Name	Airplanes on collision course
Dependencies	
Responsible	Eirik M. Hammerstad
Type	System test
Description	The user should be notified if some airplanes are on collision course.
Pre-Condition	
Execution	1. The system is started, where two planes have colliding courses.
Expected result	The system notifies the user with audio information about the airplanes and the time before the collision will occur.
Actual result	The system notified about the two planes on a colliding course, and the remaining time to the collision.
Accepted	Yes
Priority	Low

Table I.10: Test P01

ID	P01
Name	System responding
Dependencies	
Responsible	Eirik M. Hammerstad
Type	Performance
Description	The system should respond to user input in 100 ms.
Pre-Condition	All unit-tests and integration tests has passed.
Execution	1. The system is started. 2. The user sends voice or motion input to the system.
Expected result	The system responses in 100 ms.
Actual result	The system responded to user input within 100 ms
Accepted	Yes
Priority	Medium

Table I.11: Test P02

ID	P02
Name	System uptime
Dependencies	
Responsible	Joachim Halvorsen
Type	Performance
Description	The system should be able to run for 30 minutes without any errors.
Pre-Condition	All unit-tests and integration tests has passed.
Execution	1. The system is started.
Expected result	The system runs for 30 minutes without any problems.
Actual result	The system ran without any troubles for 30 minutes
Accepted	Yes
Priority	Low

Table I.12: Test P03

ID	P03
Name	Frames Per Second
Dependencies	
Responsible	Eirik M. Hammerstad
Type	Performance
Description	The system should not experience loss of performance with an average FPS below 50, when running continuously for 10 minutes.
Pre-Condition	All unit-tests and integration tests has passed.
Execution	The system is started.
Expected result	The FPS is not below 50.
Actual result	The FPS was steady on approximately 60 FPS
Accepted	Yes
Priority	Medium

Table I.13: Test P04

ID	P04
Name	15 airplanes in airspace
Dependencies	
Responsible	Eirik M. Hammerstad
Type	Performance
Description	The system should be able to run without any errors with 15 airplanes in the airspace.
Pre-Condition	All unit-tests and integration tests has passed.
Execution	1. The system is started with 15 airplanes in the airspace.
Expected result	There are 15 airplanes in the airspace and the system is running without errors.
Actual result	The system had no problems running with 15 airplanes in the airspace
Accepted	Yes
Priority	Low

Table I.14: Test U01

ID	U01
Name	Usability
Dependencies	
Responsible	Mirna Besirovic
Type	Usability
Description	The user should be able to use the system for ten minutes without making any errors after reading the manual and receiving training in the use of the system for half an hour.
Pre-Condition	All unit-tests and integration tests has passed.
Execution	1. The system is started. 2. The user executes predefined tasks provided by the testing responsible.
Expected result	The user is able to perform the given tasks in the system without any errors for ten minutes.
Actual result	The user was able to perform the given tasks satisfactorily without any errors for ten minutes
Accepted	Yes
Priority	Medium

Appendix J

Usability Testing

This chapter contains the guidelines we used conduction usability tests, the task we gave the users during the tests and the SUS and observation forms we used during testing.

J.1 Guidelines

Before conducting the usability tests we we followed the 10 point guidelines[10] that are presented below:

1. **Introduce yourself** and others involved in the test
2. **Explain** the purpose of the test
3. **Inform to participants that they can stop when they want** if they feel it is uncomfortable to continue, they do not need to explain why they want to cancel the test.
4. **Give a description of the equipment that is used** and possible limitations.
5. **Teach the user how to think out loud**, explain that they need to explain why they do the things they do in addition to what.
6. **Explain that you can not help him/her during the test**, because the goal of the test is to see how a user uses the system
7. **Describe the task and introduce the product**, but not how it works.
8. **Ask if the participant has any questions before they start**, take notes of the questions and problems the participant encounters during the test.
9. **End the test with letting the user express their views before you ask questions**, ask specific questions about the things you saw that the participant had problems with
10. **Use the results** as input for further work.

J.2 User Tasks

The tasks given to the user during usability testing are presented here.

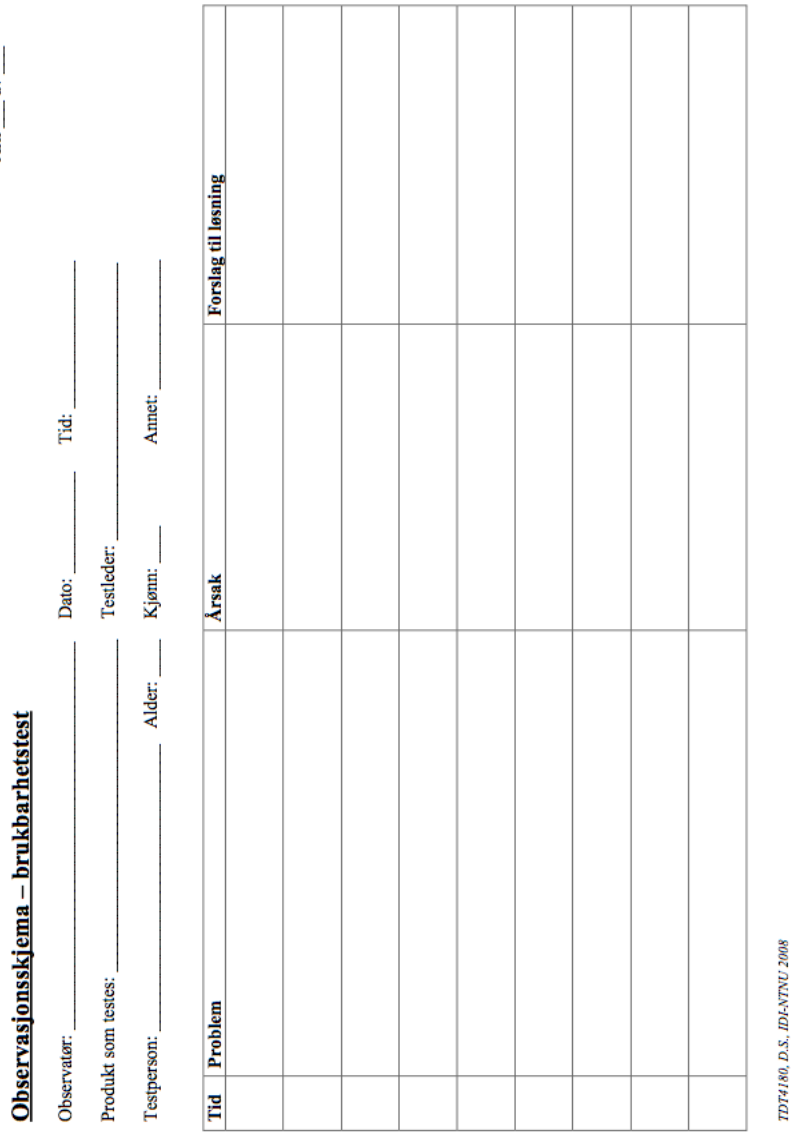
Change the camera's position and rotation.

1. You wish to take a closer look on an airplane, to see if everything is OK. Pick out an airplane yourself and select it using a predefined voice command.
2. Zoom in on the airplane using motion.

Change the airplane's directions.

1. Select airplane two.
2. Follow airplane two.
3. You wish to change airplane two's heading to west.
4. After the heading has changes you also wish to change the airplanes speed to 200 km/h
5. You wish to change the pitch of the airplane to 10 degrees
6. When the airplanes pitch has changed, you wish to follow airplane one.

J.2.1 Observation Form from Usability Testing



Observasjonsskjema – brukbarhetstest

Ark ___ av ___

Observatør: _____ Dato: _____ Tid: _____

Produkt som testes: _____ Testleder: _____

Testperson: _____ Alder: ___ Kjønn: ___ Annet: _____

Tid	Problem	Årsak	Forslag til løsning

TDT4180, D.S., IDI-NTNU 2008

Figure J.1: Observation Form[21]

Observation Form from Usability Testing

Ark 1 av 1

Observasjonsskjema – brukbarhetstest

Observatør: Mirna Besirovic   Dato: 31.10.12   Tid: 10:45

Produkt som testes: Motion and voice driven UX   Testleder: Johanne Linde

Testperson: Kristine Ystmark   Alder: 23   Kjønn: K   Annet: X

Tid	Problem	Årsak	Forslag til løsning
10:55	Er zoomet inn på annet fly enn det hun har "selected". Vet dermed ikke om flyet hun skal velge / kommando gått gjennom	Hun var ikke i nærheten av flyet hun valgte	Spesifiser i manual at man må 'follow' for å se om fly valgt + gi feedback at kommando
10:5?	Tilbakemelding om at flyet endre pitch/heading, var mangelfull	Hun visste ikke om flyet faktisk utførte kommandoen helt sikkert	Gi en form for feedback, annet enn at fly endrer seg.
10:53	Visste ikke hvilken høyde var å foretrekke for kalibrering	Ikke noe info om bra kalibreringshøyde	Kan stå bedre/mer forklart i manualen

TDT4180, D.S., IDI-NTNU 2008

Figure J.2: Observation Form from Usability Testing

Observasjonsskjema – brukbarhetstest

Observatør: Mina Besirovic Dato: 31.10.12 Tid: 10:15

Produkt som testes: Motion and voice driven UX Testleder: Johanne Linde

Testperson: Hege Beate Seilen Alder: 22 Kjønn: K Annet: X

Tid	Problem	Årsak	Forslag til løsning
	Misforstå hvor kalibrering av linjen skulle gjøres	Mulig for dårlig forklaret i manual	Kan stå bedre/mer forklaret i manualen
	System (voice recognition) oppfatter ikke alle kommandoene hennes	British uttale	Spesifiser bedre i manual et amerikansk uttale
	Går seg "vill" i blå himmel	Kan hende kamera har havnet "under bakkenivå"	Lage begrensning - kamera kan ikke gå lenger enn punktet
	Dårlig tilbakemld. Vet ikke om endring skjer.	Ingen feedback på om kommando har blitt registrert	Gi en form for feedback annet enn at flyet endrer seg

Figure J.3: Observation Form from Usability Testing

J.2.2 SUS Form from Usability Testing

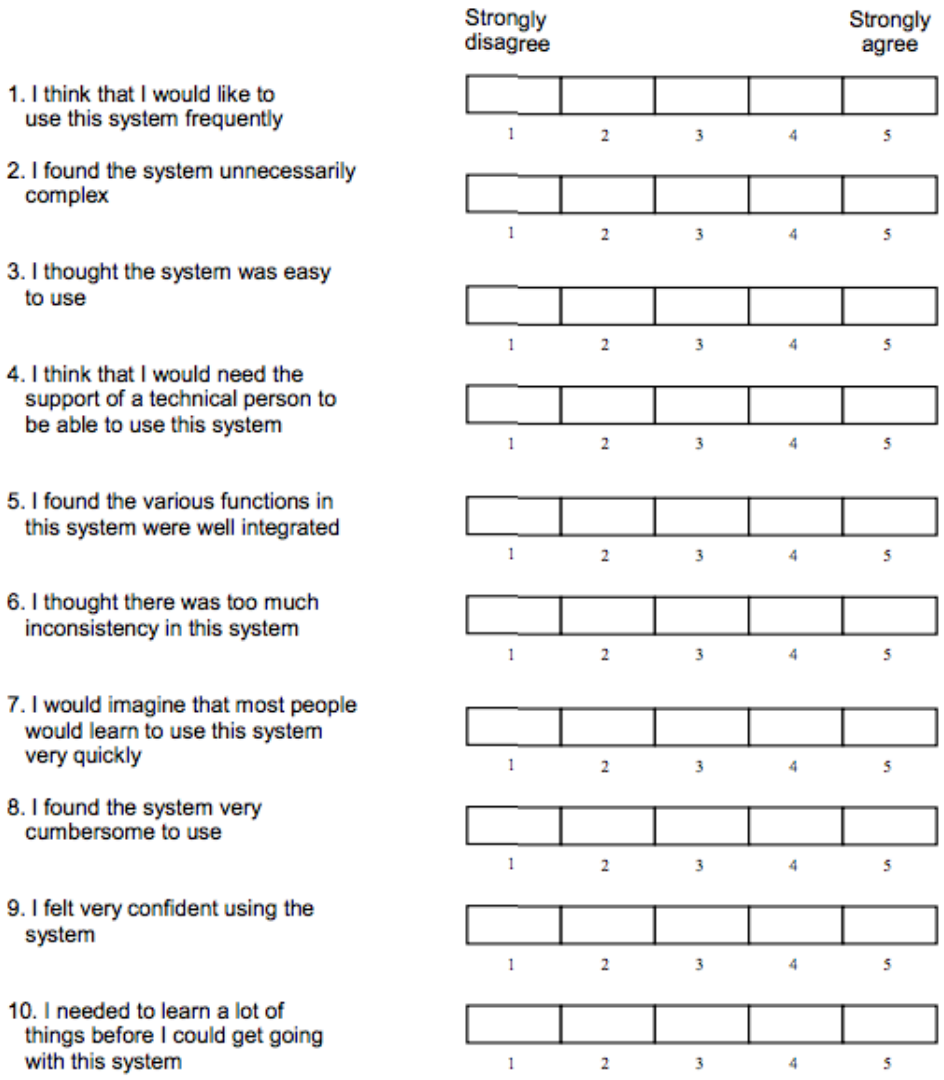
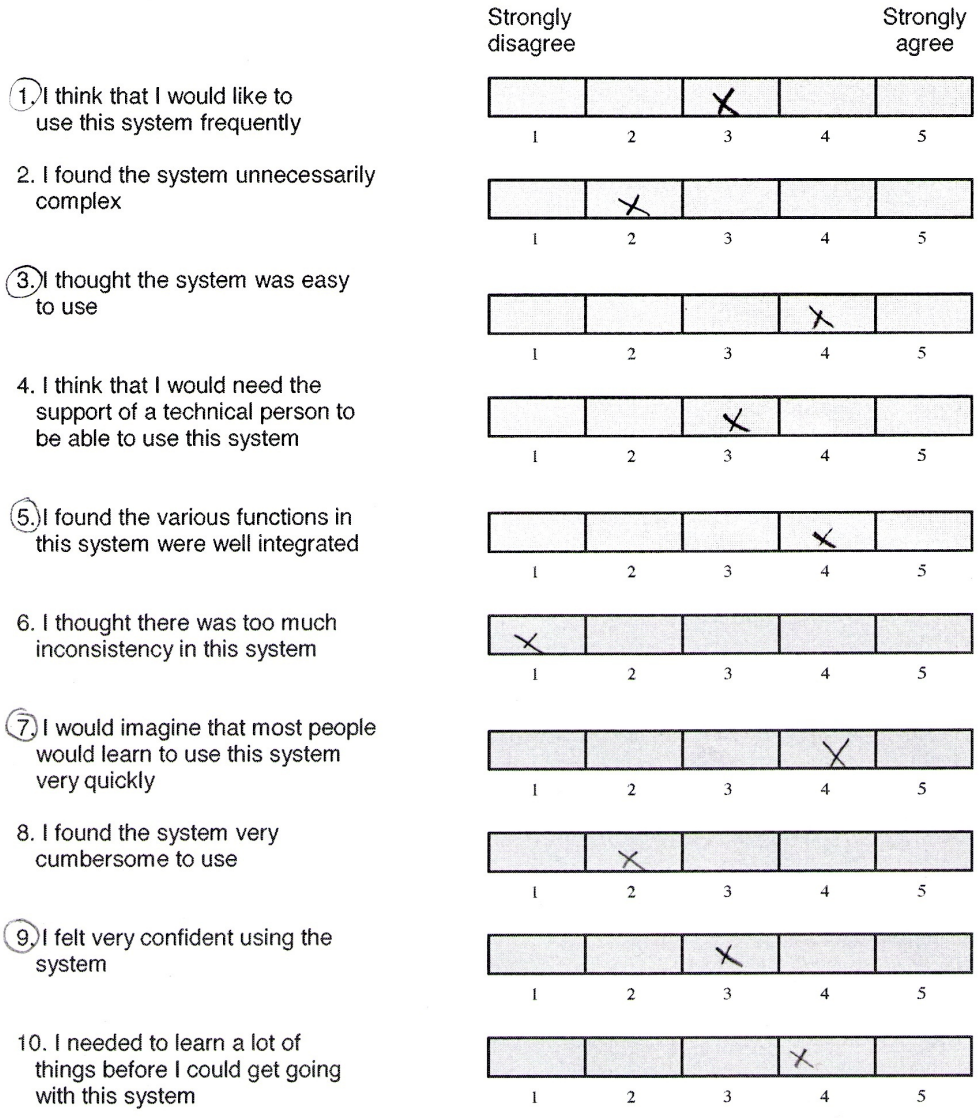


Figure J.4: System Usability Scale[24]

J.2.3 SUS Form filled by user 1

System Usability Scale

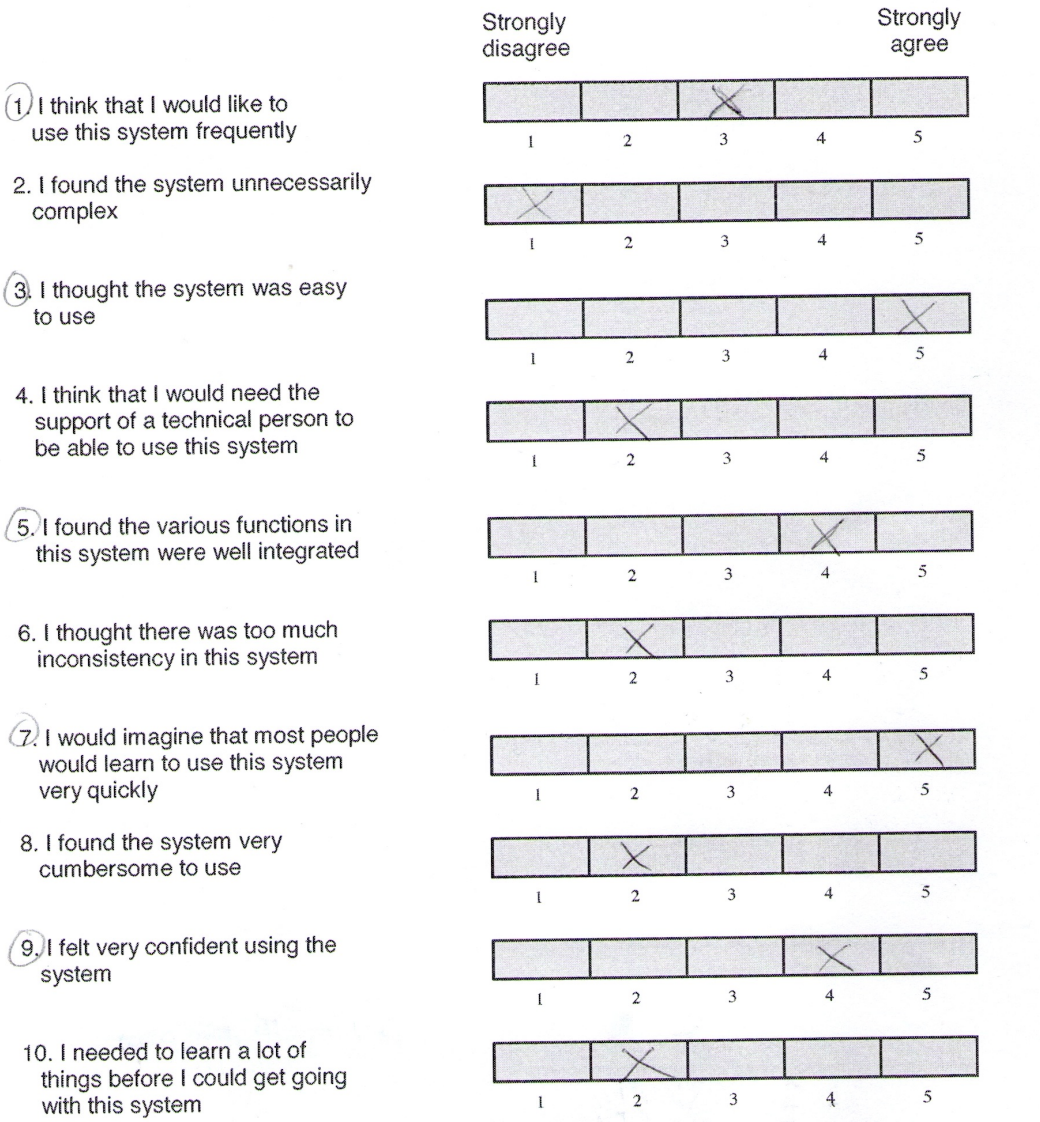


→ 2 + 3 + 3 + 3 + 2 = 13
→ 3 + 2 + 4 + 1 + 3 = 13
} 26 · 2,5 = 65

Figure J.5: System Usability Scale from user 1

J.2.4 SUS Form filled by user 2

System Usability Scale



→ 2 + 4 + 3 + 4 + 3 = 16
→ 4 + 3 + 3 + 3 + 3 = 16
} 32 · 2,5 = 80

Figure J.6: System Usability Scale from user 2

Appendix K

Manual

State: The program is running with a Kinect device attached to the computer.

K.1 Voice commands

Select airplane:

Say: "Select plane" + <Plane ID>
The airplane is marked red, indicating that it has been selected.

Change heading of selected airplane:

Say: "Alter heading" + <North/South/East/West/Angle>
(North = 0 degrees)

Change pitch of selected airplane:

Say: "Alter pitch" + <Angle>

Change speed of selected airplane:

Say: "Alter speed" + <Speed>

Deselect a selected airplane:

Say: "Deselect"
The previously selected plane is no longer marked red, and no longer selected.

Toggle whether or not the names of the planes are displayed:

Say: "Toggle names"

Toggle whether or not the coordinates of the planes are displayed:

Say: "Toggle coordinates"

Toggle whether or not the lines below the planes are displayed:

Say: "Toggle lines"

Toggle whether or not the help-text is displayed:

Say: "Toggle help"
Possible commands are then displayed

Toggle whether or not voice feedback is given.

Say: "Toggle voice"

Zoom in on the selected airplane:

Say: "Go to"

Follow the selected airplane:

Say: "Follow"

Stop following an airplane:

Say: "Unfollow"

K.2 Motion commands

Change camera position:

- Hold left hand above the line on the Kinect video stream.
- Move the left hand forward/backwards to move camera forward/ backwards (Zoom in/out)

Change camera rotation:

- Hold right hand over the line on the Kinect video stream.
- Move the right hand forward/backwards to rotate camera down/up

Zoom in

Figure K.1: Zoom In

Zoom out

Figure K.2: Zoom Out

- Move the left hand to the side to move camera to the same side

Figure K.3: Move Camera position

- *Rotate down*

Figure K.4: Rotate Camera Down

- *Rotate up*

Figure K.5: Rotate Camera Up

- Move the right hand sideways to rotate camera to the same side.

Figure K.6: Rotate Camera to the Side

"Pause"

- For pausing motion detection hold both hands anywhere below shoulder-height.

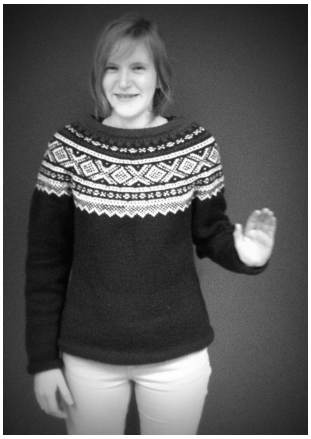


Figure K.7: Pause

Appendix L

Time Spent

The time spent on the project in each week is shown in Table L.1. The last project week, week 47, is not shown here. The reason for that is that we sent the report for printing before this week. In week 47, we planned to only work with the presentation.

The time spent on the different tasks is shown in Table L.2.

Table L.1: Time spent on the project in each week

Week	**Hours**
Week 34	102
Week 35	126
Week 36	126
Week 37	124
Week 38	121
Week 39	130
Week 40	122
Week 41	120
Week 42	119
Week 43	119
Week 44	135
Week 45	129
Week 46	134
Total	**1607**
Average per person per week	**24.7**

Table L.2: Time spent on each task

Task	**Hours**
Lectures and self study	95
Project management	140
Meeting	96.5
Planning	105.5
Pre-study	116.5
Requirement specification	48
Programming	337
Documentation	574
Architecture	21.5
Testing	55
Presentation	18
Total	**1607**

Bibliography

[1] Conradi R., Gulla J. A., Ingvaldsen J. E., Hagen S., Solskinnsbakk G., Landmark A. D., Duc A. N., Petersen S. A. *Compendium: Introduction to course TDT4290 Customer Driven Project, Autumn 2012*, 2012

[2] Len Bass, Paul Clements, Rick Kazman, *"Software Architecture in Practice, Second Edition"* , Addision-Wesley, 2003.

[3] . Travis Swicegood, Chapter 1, *Pragmatic Version Control Using Git* , 2008

[4] Y. Daniel Liang, Chapter 1. *Introduction to Java Programming*, seventh edition, 2009.

[5] Bjarne Stroustrup, Chapter 1, *The C++ Programming Language*, Third Edition, 1997.

[6] Kruchten P, *Software Architecture in Practice* "Architectural Blueprints - The "4+1" View Model of Software Architeture", 1995.

[7] Rrvik C.J., Emanuelsen E., Fosse H, Geithus H, Stenersen S, Gamnes V., *Spatial Gesture Investigation*, "How to select" p. 78, 2011.

[8] Hass J., Weir M.D. and Thomas G.B, section 1.3, page 24, "Trigonometry Functions", *Calculus 1 TMA4100*, Pearson Custom Publishing, 1997.

[9] Ian Sommerville, "Waterfall" page 73-74, *Software Engineering 2. edition*, Addison-Wesley, 2010.

[10] Rogers, Sharp and Preece, "10 Guidelines" *Interaction design - beyond human-computer interaction. 3rd edition*, 2011

[11] Truls Thirud, Computas AS, 11.06.2012, Presentation.

[12] Wang A. I, TDT4240, NTNU. *Video Games and Software Architecture*, slide 23, 30/01/2012.

[13] Wang A. I, TDT4240, NTNU. *Achieving Qualities: Quality Tactics*, 23/01/2012

[14] Srensen C.-F.,TDT4240, NTNU. *Design Patterns*, 02/02/2012

[15] Reidar Sande, BEKK Consulting, *Presentation on agile development and estimation*, 03/09/2012

[16] Tor Stålhane, TDT4242 *Requirements and Test. Presentation on White Box vs Black Box Testing*, 23/02/2012

[17] Tor Stålhane, TDT4242 *Requirements and Test. Presentation on Grey Box Testing*, 25/02/12

[18] Tor Stålhane, TDT4242 *Requirements and Test. Presentation on White Box Testing*, 23/02/12

[19] Tor Stålhane, TDT4242 *Requirements and Test. Presentation on Writing a Test Strategy*, 25/03/12

[20] Tor Stålhane, TDT4242 *Requirements and Test. Presentation on Test Driven Development*, 25/03/12

[21] Dag Svanæs, TDT4180 IDI, NTNU. *Observasjonsskjema - brukbarhetstest*, 2008.

[22] The Institute of Electrical and Electronics Engineering, Inc. *IEEE 829 Standard for Software and System Test Documentation*, 18/07/2008

[23] ISO 9241-11, *ergonomics of human-computer interaction*, 1998

[24] John Brooke, *SUS - A quick and dirty usability scale*

[25] Henrik Pryser Libell , Ellen S. Viseth and Mlfrid Bordvik. "Air traffic controller shortage gives delays at Gardemoen". *VG* 10/07/12. Accessed 10/07/12.

`http://www.vg.no/reise/artikkel.php?artid=10066582`

[26] "What is Scrum", *Mountain Goat Software* Accessed 21/09/2012.

`http://www.mountaingoatsoftware.com/topics/scrum`.

[27] Paul Hoadley, "Waterfall Model", *Wikipedia* 25/11/2005. Accessed 21/09/2012.

`http://en.wikipedia.org/wiki/File:Waterfall_model.png`.

[28] "Waterfall model" *Wikipedia* 25/02/2002. Accessed 29/08/2012.

`http://en.wikipedia.org/wiki/Waterfall_model`

[29] Excirial, "A quick overview of the Test-driven development lifecycle." *Wikipedia* 12/05/2009. Accessed 21/09/2012.

`http://en.wikipedia.org/wiki/File:Test-driven_development.PNG`.

[30] "Kinect ver. 1.5 coming in May, with speech recognition and seated tracking". *GMA network* 28/03/2012. Accessed 28/09/2012.

`http://www.gmanetwork.com/news/story/253080/scitech/gaming/kinect-ver-1-5-coming-in-may-with-speech-recognition-and-seated-tracking`.

[31] "Download GanttProject." *Gantt Project*. Accessed 28/08/2012.

`http://www.ganttproject.biz/download`.

[32] "Violet UML editor." Accessed 04/09/2012.

`http://alexdp.free.fr/violetumleditor/page.php`.

[33] "What are Google Docs, Sheets and Slides?" *Google*. Accessed 28/08/2012.

`http://support.google.com/drive/bin/answer.py?hl=en&answer=49008`.

[34] *Google Docs/Drive*. Accessed through the whole project period.

`www.docs.google.com` `www.drive.google.com`

[35] Tan, K. "25 (Free) 3D Modelling Applications You Should Not Miss". *Hongkiat*. Accessed 25/09/2012.

`http://www.hongkiat.com/blog/25-free-3d-modelling-applications-you-should-not-mi`

[36] *Blender*. Accessed 04/09/2012.

`http://www.blender.org/`

[37] "Microsoft XNA Game Studio 4.0" *Microsoft*. Accessed 04/09/2012.

`http://www.microsoft.com/en-us/download/details.aspx?id=23714`

[38] "Microsoft XNA Framework Redistributable 4.0" *Microsoft*. Acessed 04/09/2012.

`http://www.microsoft.com/en-us/download/details.aspx?id=20914`

[39] "Microsoft XNA" *Wikipedia* 27/03/2004. Accessed 24/09/2012.

`http://en.wikipedia.org/wiki/Microsoft_XNA`

[40] Plunkett Luke, "Report: Here Are Kinects Technical Specs" *Kotaku*. Accessed 24/09/2012

`http://kotaku.com/5576002/here-are-kinects-technical-specs`

[41] *Skype*. Accessed 23/08/2012.

`www.skype.com`

[42] *Facebook.* Accessed 23/08/2012.

www.facebook.com

[43] *Teamviewer.* Accessed 23/08/2012.

http://www.teamviewer.com/en/index.aspx

[44] *Balsamiq.* Accessed 14/09/2012.

http://www.balsamiq.com

[45] *Gimp.* Accessed 04/09/2012.

http://www.gimp.org/.

[46] Jonathan Wolter, Russ Ruffer, Misko Hevery, "Writing Testable Code, Google tech talks" *Misko Hevery* Accessed 01/09/2012.

http://misko.hevery.com/attachments/Guide-Writing%20Testable%20Code.pdf

[47] "Flight dynamics", *Wikipedia* 13.12.2010. Accessed 27/08/2012.

http://en.wikipedia.org/wiki/Flight_dynamics.

[48] "Scrum (development)", *Wikipedia* 14/04/2006. Accessed 28/08/2012.

http://en.wikipedia.org/wiki/Scrum_%28development%29

[49] "3 Key Scrum Benefits", *Accelerright* Accessed 29/08/2012.

http://www.accelright.com/agile-scrum/3-key-scrum-benefits/

[50] "Scrum Retrospective" *Mountain Goat Software* Accessed 29/08/2012, from

http://www.mountaingoatsoftware.com/scrum/sprint-retrospective

[51] Tang, Matthew, "Hand Gestures Recognition Using Microsoft's Kinect", *Stanford* 16/03/2011. Accessed 21/08/2012.

http://www.stanford.edu/class/ee368/Project_11/Reports/Tang_Hand_Gesture_Recognition.pdf

[52] "iisu SDK", *Soft Kinetic.* Accessed 05/10/2012.

http://www.softkinetic.com/en-us/solutions/iisusdk.aspx

[53] "Kinect Getting Started Become The Incredible Hulk", *Microsoft Blogs* 17/06/2011. Accessed 05/10/2012.

http://blogs.microsoft.co.il/blogs/shair/archive/2011/06/17/kinect-getting-started-become-the-incredible-hulk.aspx

[54] "Setting up your Development Environment', *Microsoft Blogs* 17/06/2011. Accessed 05/10/2012.

http://channel9.msdn.com/Series/KinectQuickstart/Setting-up-your-Development-Environment/player?w=512&h=288

[55] "Facts about Itera Consulting", *Itera Consulting*. Accessed 25/08/2012.

http://www.iteraconsulting.no/Om-oss/Itera-Consulting-pa-1-minutt/

[56] "The Future of Smart TV, Now", *Samsung*. Accessed 05/09/2012.

http://www.samsung.com/sg/smarttv/

[57] "Siri (software)", *Wikipedia*, 03/12/2011. Accessed 05/09/2012.

http://en.wikipedia.org/wiki/Siri_(software)

[58] "Kinect sports", *Xbox* Accessed 05/09/2012.

http://marketplace.xbox.com/en-US/Product/Kinect-Sports/66acd000-77fe-1000-9115-d8024d5308c9

[59] "Kinect sports picture", *Xbox* Accessed 05/09/2012.

http://kotaku.com/5679411/review-kinect-sports

[60] Clifton Marc, "Advanced Unit Testing, Part I - Overview" *Code Project*, 18/11/2008. Accessed 13/11/2012.

http://www.codeproject.com/Articles/5019/Advanced-Unit-Testing-Part-I-Overview

[61] "Integration testing." *Wikipedia*. Accessed 27/10/2012

http://en.wikipedia.org/wiki/Integration_testing

[62] "System testing", *Wikipedia* 28/07/2004. Accessed 27/10/12.

http://en.wikipedia.org/wiki/System_testing

[63] "Functional testing", *Wikipedia* 04/08/2006. Accessed 02/10/12.

http://en.wikipedia.org/wiki/Functional_testing

[64] "An Introduction to Kinect", *Microsoft*. Accessed 25/09/12.

http://www.microsoft-press.de/productinfo.asp?replace=false&cnt=productinfo&mode=2&type=2&id=mp-6396&index=2&nr=0&sid=6fd99a5542f17cb1ff480d3db9f6f45c&preload=false&page=1&view=fit&Toolbar=1&pagemode=none

[65] Leigh, Alexander. "Microsoft: Kinect Hits 10 Million Units, 10 Million Games.", *Gamasutra* 09/03/2011. Accessed 25/09/12.

`http://www.gamasutra.com/view/news/33430/Microsoft_Kinect_Hits_10_Million_Units_10_Million_Games.php#.UI2Kn8U4ld8`

[66] "Leap Motion: Kinect Style 3D Gesture Based Hands Free PC Control," *ITechWiz*. Accessed 25/09/12.

`http://www.itechwhiz.com/2012/05/leap-motion-kinnect-3d-pc-controller.html`

[67] Plunket, Luke. "Report: Here are the Kinect's Technical Specs". *Kotaku* 30/06/2010. Accessed 25/09/12.

`http://kotaku.com/5576002/here-are-kinects-technical-specs`

[68] "Microsoft Kinect for Windows SDK - V1.0 Release Notes." *Microsoft* 02/05/2012. Accessed 06/11/12.

`http://www.microsoft.com/en-us/kinectforwindows/develop/release-notes.aspx`

[69] "Freeze (software engineering)". *Wikipedia* 05/05/2005. Accessed 01/10/12.

`http://en.wikipedia.org/wiki/Freeze_%28software_engineering%29`

[70] "How does Planning Poker work?" *Mountain Goat Software.* Accessed 29/09/12.

`http://www.planningpoker.com/`

[71] "Java (programming language)." *Wikipedia* 13/11/2001. Accessed 21/09/12.

`http://en.wikipedia.org/wiki/Java_(programming_language)`

[72] C Sharp (programming language)" *Wikipedia* 25/10/2001. Accessed 21/09/12.

`http://en.wikipedia.org/wiki/C_Sharp_(programming_language)`

[73] "Open Kinect" *OpenKinect*. Accessed 21/09/2012.

`http://openkinect.org/wiki/Main_Page`

[74] "Official Kinect SDK vs. Open-source alternatives". *Stack Overflow* Accessed 21/09/2012

`http://stackoverflow.com/questions/7706448/official-kinect-sdk-vs-open-source-alternati`

[75] "Moq, the simplest mocking library for .NET and Silverlight". *Google code*. Accessed 29/09/2012.

`http://code.google.com/p/moq/`

[76] "Fakenect" *OpenKinect*. Accessed 12/10/2012.

`http://openkinect.org/wiki/Fakenect`

[77] "MocKinect." *OpenKinect*. Accessed 12/10/2012.

`http://mockinect.codeplex.com/`

[78] "Speech Recognition for the Kinect, the Easy Way..." *blogspot*, 16/06/2012. Accessed 12/10/12.

`http://kin-educate.blogspot.no/`

[79] "Git (Version Control)". *Wikipedia*, 20/05/2005. Accessed 26/09/2012.

`http://en.wikipedia.org/wiki/Git_(software)`

[80] "Team Foundation Server." *Microsoft*. Accessed 26/09/12.

`http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx`

[81] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato, "Version Control with Subversion: For Subversion 1.4" Chapter 1. Accessed 26/09/12.

`http://svnbook.red-bean.com/en/1.4/svn-book.pdf`

[82] "Hardware and Operating System Requirements", *Microsoft*. Accessed 12/11/2012.

`http://msdn.microsoft.com/en-us/library/bb203925.aspx`

[83] "Microsoft Kinect for Windows SDK V1.0 release notes." *Microsoft*, 02/05/2012. Accessed 12/11/2012.

`http://www.microsoft.com/en-us/kinectforwindows/develop/release-notes.aspx`

.