NTNU

Department of Computer and Information Science

TDT4290

Customer driven project

BoligApp

Group 6:
Nenad Milosavljevic
Dusan Stefanovic
Jørgen Faret
Johan Reitan

Supervisor:
Anh Nguyen Duc

Customer:
Eli Nordvik
Elin Svendsen
Lars Kristian Sundset

November 20, 2013

NTNU
Det skapende universitet

Abstract

The course TDT4290 Customer Driven Project is a master-level course hosted by the department of computer and information science at NTNU. The purpose of the course is to give computer science students an introduction to software development in the real world and the problems and challenges this entails. The objective of the project is to make a realistic prototype of an information system for a 3rd party customer.

In this case, the customer is NAV Hjelpemiddelsentral Sør-Trøndelag. NAV (Norges Arbeids- og Velferdsetat) is the governmental welfare agency in Norway, and NAV Hjelpemiddelsentral is the branch of NAV that deals with assisting people with disabilities in Norway.

The development team consists of four students at NTNU, the Norwegian University of Science and Technology. The students are taking this course as a part of their master's degree in computer science.

The task specified by the customer was to create a mobile application for iOS to assist in the planning of room layouts when modifying houses to facilitate the installation of assistive technology for people with disabilities.

This report contains a documentation of the development process, from preliminary study and planning through implementation and finally a project summary and evaluation.

Preface

We would like to thank our supervisor, Anh Nguyen Duc, for his guidance throughout this course. We would also like to give a special thank you to our customer contacts Eli Nordvik, Elin Svendsen and Lars Kristian Sundset for their enthusiasm as well as their eagerness to assist us with any problems we encountered throughout the course of the project.

Contents

List of Figures

List of Tables

Part I

Planning and project management

Chapter 1

Project directive

1.1 Project name

The name of the project is BoligApp. This is also going to be the name of the application to be developed.

1.2 Project mandate

The purpose of the project is to develop an application to assist in the planning of room layouts when modifying houses to facilitate the installation of assistive technology for people with disabilities. The application should ensure the plan is in accordance with Norwegian building laws and regulations. In addition, the application should be able to connect to NAV's assistive technology database and import items from it into the plan.

1.3 Project duration

The development team was created and first customer contact established Wednesday the 21st of August. The final report of the project is to be delivered and presented on Thursday the 21st of November. The group has therefore set the deadline for finishing the project to Sunday the 17th of November.

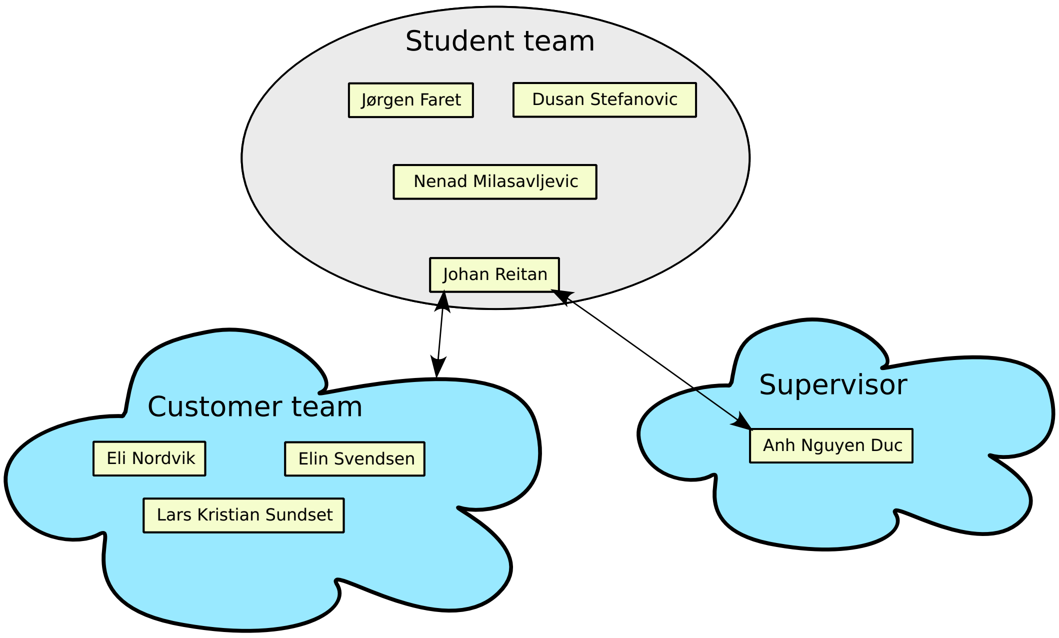

Figure 1.1: Illustration of the project stakeholders, and their communication usually through Johan Reitan, the communication responsible

1.4 Project stakeholders

Customer	
Company name	NAV Hjelpermiddelsentral Sør-Trøndelag
Customer contacts	
Eli Nordvik	`eli.nordvik@nav.no`
Elin Svendsen	`elin.svendsen@nav.no`
Lars Kristian Sundset	`lars.kristian.sundset@nav.no`
Development team	
Johan Reitan	`joharei@stud.ntnu.no`
Jørgen Faret	`jorgenfa@stud.ntnu.no`
Dusan Stefanovic	`dusans@stud.ntnu.no`
Nenad Milosavljevic	`nenadm@stud.ntnu`
Advisor	
Anh Nguyen Duc	`anhn@idi.ntnu.no`

1.5 Customer description

NAV is the governmental welfare agency in Norway. NAV Hjelpemiddelsentralen Sør-Trøndelag (NAV HMS) is the branch of NAV that deals with assisting people with disabilities in the region Sør-Trøndelag in Norway. They have a lot of assistive technology that they need to install into the houses of their patients. Today, the employees at NAV HMS need to draw plans of the houses by hand in order to visualise how they will look like, and if they comply with the laws and regulations. This is tedious work, and could be done easier and more accurate with the use of an app like BoligApp.

Chapter 2

Project plan

2.1 Project phases

On the first week members should get to know each other and familiarize themselves with the project. The first meeting with the customer is also held on the introductory week. After we get all necessary information from the customer, we are going into the project planning phase (2.1.1). This phase lasts for two weeks in which we are going to focus on risk assessment, understanding the specifications of the project and creating a WBS. In this phase, all administrative tasks should be done, including getting the required equipment from the customer in order to develop. This phase will overlap with the next one where we are going to focus on learning the skills needed for finishing the project. Our main goals would be to get comfortable with iOS application development, graphical design, computer vision and databases in general. Actual development is going to start when the second phase is over. It is going to include 4 sprints and one introductory sprint, and will last for 7 weeks. The main product of the development phase is a complete application that we can present to the customer. The testing phase is going to occur next. Unit testing, integration, system and final acceptance testing will be done in this phase. The last three weeks are reserved for finishing the documentation and the project report, which will also be written throughout the whole process. During this period, the remaining sections should be completed, the whole document thoroughly revised and prepared for final delivery.

2.1.1 Planning and research

In this phase we are going to introduce each other to the course, do basic planning of the project, make a list of requirements, role distribution and preliminary study.

2.1.2 Learning

Our main focus here will be on learning the Objective C programming language, as the team is not familiar with it, and developing iOS applications in general. Besides that, we are going to learn about databases, 2D modelling and computer vision. This phase is crucial for a successful developing phase.

2.1.3 Sprints

We have decided to use the scrum methodology for this project, which means our work will be divided into sprints. For more information about our choice of development methodology, see section 3.1.4.

Before beginning a new sprint, a sprint planning meeting should be held. The scrum master, the rest of the team and the customer should attend this meeting, in which they are going to choose which features and functionalities have the highest priority. Based on that information, the chosen task is moved from the product backlog to the sprint backlog. The scrum team and the customer should collectively define a sprint goal which is later discussed during the sprint review meeting. See Figure 2.1 for an illustration.

2.1.4 Developing

In this phase our focus will be on actual coding. Code refactoring, and other code fixing tasks will be done in this phase.

2.1.5 Testing

The testing phase represents time spent on testing the system. Integration, unit testing, system testing, etc. should be done here. This phase is going to overlap with developing phase due to test-driven development methodology.

2.1.6 Documentation

The last part of the project will include evaluation of our work, finishing documentation and final presentation for advisor and customer.

2.2 Milestones

We have split the development of our project into four main segments. Each milestones will be covered by one sprint.

2.2.1 Milestone 1 - Load and display plan

The first functionality to be implemented is the portion of the program that enables the application to load and display a plan. This is a core functionality for the application and implementing this will be the first milestone.

2.2.2 Milestone 2 - Make plan interactive

The second main functionality to be implemented is make the plans interactive. This entails being able to move, resize, rotate and remove items in the plan correctly.

2.2.3 Milestone 3 - Expand plan functionality

The third main functionality to be implemented is expand the functionality of the application by adding the ability to draw new plan elements in addition to adding functionality of the plan viewer by adding the ability to zoom and pan in the plan.

2.2.4 Milestone 4 - Import items from database

The fourth main functionality to be implemented is to connect to and load items from the assistive technology database NAV has. This also entails creating an interface that translates attributes from the database into graphical attributes in our plan.

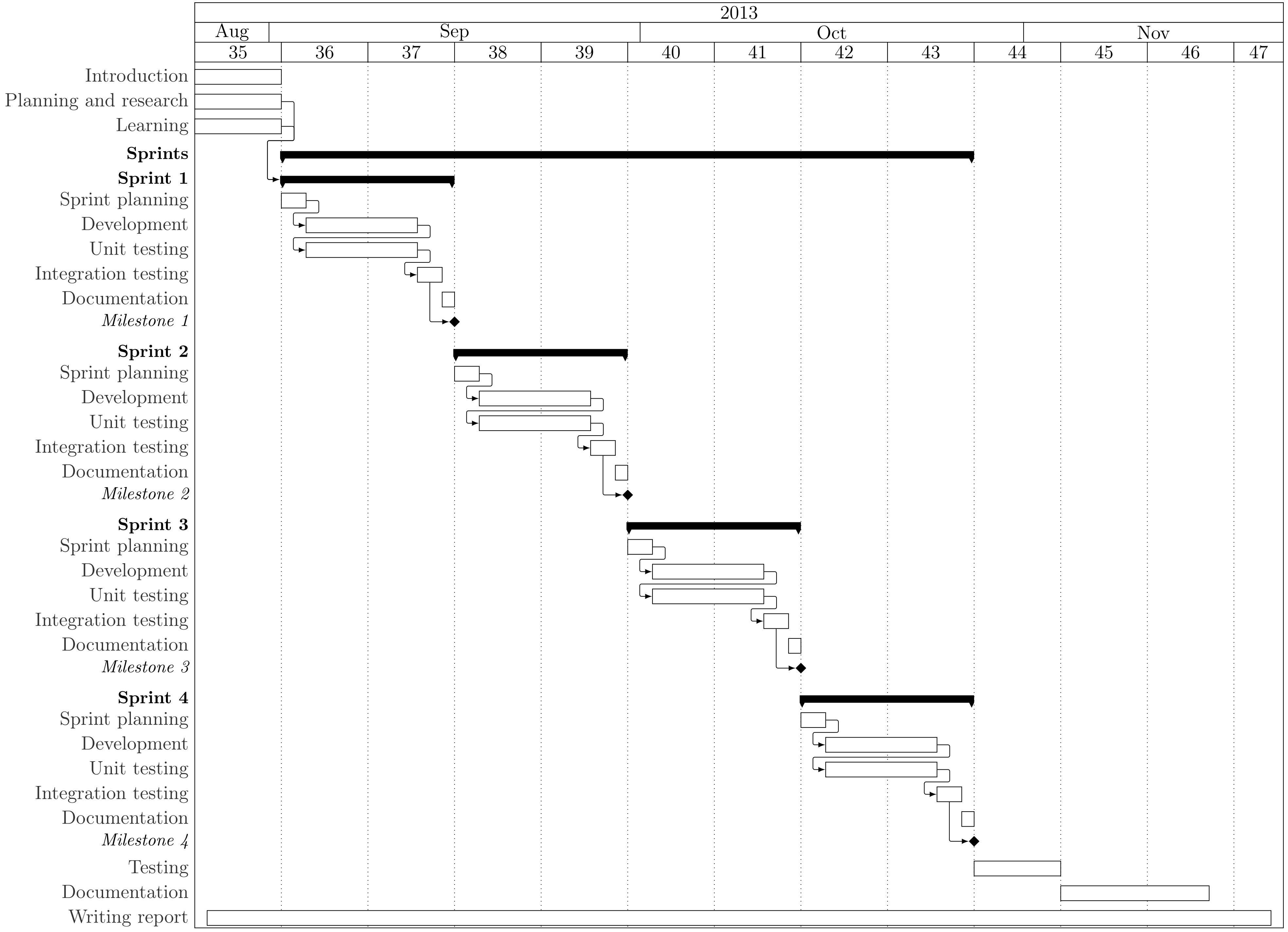

Figure 2.1: Gantt diagram illustrating the project plan

2.3 Risk assessment

Table 2.1 shows a list of possible problematic situations that could occur during the project work.

Table 2.1: Table of risks

Risk description	*Probability*	*Severity*	*Consequences*	*Preventive actions*
Unfamiliar/ inexperienced with development platform	H	M	Delays, lesser product quality	Read documentation, follow tutorials, ask experts
Poor time planning	H	L	Overtime, delays, lesser product quality	Use generous time estimates in the beginning. Update estimates continuously as the project progresses, and new problems surface
Customer changes their minds	H	M	Unnecessary work (work that is already finished could be discarded), delays, overtime	Keep up good communication with the customer to get to know the change of plans as early as possible
Customer does not have technical expertise	H	H	Difficult to get the required information, delays	Early contact with the customer in order to find out where the required information can be found.

Continued on the next page

Risk description	*Probability*	*Severity*	*Consequences*	*Preventive actions*
Missing equipment	M	H	Delays, functionality not implemented	Talk to the customer about acquiring necessary equipment. Make the best of what we have
The scope of the project is too big	H	M	Project will not meet the specifications	Take time to plan properly, so that we can estimate how much of the specification we are actually able to deliver, and report to the customer as soon as possible
Unexpected team member absence	M	H	Overtime, delays. The person absent could miss much of the project development and have a hard time catching up when he comes back	Keep the person concerned updated on the progress. Make sure all project material is available for everyone at all times
Loss of data	L	H	Varying, but could be fatal	Backup and version control system (git, Google Drive and Dropbox), be careful
Difficult problems during implementation etc.	H	M	Overtime, delays, change of plans	Be well prepared (use the learning phase (2.1.2) effectively), research problem, ask for help

Continued on the next page

Risk description	*Probability*	*Severity*	*Consequences*	*Preventive actions*
Technical problems with tools and software	M	M	Overtime, delays, loss of data, frustration and demoralization, change of plans	Choose software carefully, ask for help
Internal conflicts	L	M	Lower work morale and productivity, delays	Make sure everyone has their say, be involved in the project, avoid unequal distribution of work

2.4 Roles

Each role assignment with exception of the Scrum Master is fixed throughout the duration of the project. The Scrum Master role is dynamic, and every team member will be Scrum Master for one of four sprints.

Scrum Master - Dynamic The scrum master is responsible for keeping track of team progress, observing individual progress and making sure that every task is completed on time and managing workload. He is also responsible for the weekly status reports and ensuring that the process is followed, including issuing invitations to daily scrums, sprint reviews, and sprint planning. He is also responsible for preparing meeting agendas and managing scrum meetings.

Communication Responsible - Johan Reitan The communication responsible manages communication with external stakeholders. His main duty is to establish and maintain good communication between the team and the customer.

Development Responsible - Dusan Stefanovic The development responsible is responsible for design and implementation of the product. He has to manage architecture and code reviews and the progress of the application development. He is also responsible for code standards and conventions, code quality and code documentation.

Documentation Responsible - Johan Reitan The documentation responsible is responsible for keeping the documentation up-to-date with the development and for putting the project report together. He has to manage documentation reviews and to have an overview of the entire document, how it is structured, and to maintain a steady progress.

Team Leader - Dusan Stefanovic The team leader has the responsibility to organize and manage the team, resolve conflicts, and lead internal meetings. He should also ensure that members are motivated and that everybody gets proper tasks.

Test Responsible - Jørgen Faret The test responsible is the person responsible for creating and making sure tests are run. He should create standards for testing, such as unit tests, ensure that the unit tests are up to date, and cover an adequate amount of functionality. He is also responsible for creating integration tests, usability tests and any other tests deemed necessary for the completion of the project.

Quality Assurance Responsible - Nenad Milosavljevic The QA responsible is responsible for ensuring that the team is following the routines and standards agreed upon throughout the project in regards to group interactions and meeting routines.

Software Repository Responsible - Johan Reitan The software repository responsible helps the team members learn how to use the repository system agreed upon. In addition he is also responsible for monitoring the repository and ensuring that it is being used correctly.

Secretary - Jørgen Faret The secretary is responsible for taking minutes from all meetings with the customer and the supervisor. He is also responsible for setting up team meetings and booking rooms.

2.5 Test plan

This section will describe the test plan for the project. The test plan specifies the a description of the test methods to be used, a description of the procedures when performing the different types of tests, as well as a formal layout to be used when documenting tests.

The purpose of testing is to find potential bugs and errors in the code so that these may be corrected. In addition, during the higher levels of testing when the application is completed we will perform an evaluation of the system in accordance with the ISO-9126 standard for evaluation of software quality.

2.5.1 Testing techniques

There are three main techniques for software testing: black-box testing, gray-box testing and white-box testing.

White-box testing

White-box testing is a test technique that tests internal workings of an application as opposed to its functionality. In white-box testing, the tester is fully aware of all the implementation details of the program, and the tests are typically performed by the developer.

Although this testing technique is very useful for uncovering bugs in functions or methods, it is not very well suited for detecting unimplemented parts of the specification or missing requirements. White-box testing is applicable to testing on several levels from early to middle phases of the development cycle. In practice it is most commonly used to perform unit testing.

Black-box testing

Black-box testing is a testing technique that tests the functionality of an application, without peering into the application's internal structure. In this form of testing, the application can be seen as a "black box" where user input is the input and application behavior and response is the output. No knowledge of the internal structure and design of the program is required when performing black-box testing, which means this testing technique can be performed by anyone from developers to end-users.

Black-box testing is applicable to all levels of testing, but is well suited for and typically used to perform high-level testingat the end of the development phase such as scenario testing and final acceptance testing.

Gray-box testing

Gray-box testing is a combination of black-box testing and white-box testing. While in black-box the internal structure of the application is completely unknown to the taster and in white-box the internal structure is completely known to the tester, in gray-box the internal structure is partially known to the tester. This means that the tester has access to internal data structures and algorithms for purposes of designing test cases, but the actual testing is performed at the black-box level.

Gray box testing is considered to be non-intrusive and unbiased because it does not require the tester to have access to source code. This form of testing is typically used to perform integration testing at the mid stages of the development phase.

2.5.2 Testing methods

We will use four different testing methods throughout the development of our application: unit testing, integration testing, system testing and scenario testing.

Unit testing

Unit testing is a low-level testing method based upon the concept of verifying that individual units of code work as expected. Usually the programmer doing the coding is the same person that writes the tests, and unit testing is often performed continuously with the development of the program. Tests are typically very simple, such as for example executing a function and printing the input and output.

Integration testing

Integration testing is a natural extension of unit testing. After the individual units have been created and tested, the units are combined and these units along with the interfaces that are used to connect them are tested.

System testing

System testing is a high-level testing method that is conducted on a complete system to evaluate that the system works properly and is in accordance with the requirements for the system. System tests test both design and behavior of the system.

Scenario testing

Scenario testing is a testing method that gives a user simulates scenarios and asks the user to use the program given the conditions of the scenario. Ideally, these scenarios will portray a realistic use-case of the program and have an outcome that is easy to evaluate.

2.5.3 Testing approach

In the development of our application, we will use all of the aforementioned testing methods and techniques. We will continuously perform unit testing while writing code for each individual class during the sprints. This will be performed using white-box testing. Because these unit tests will be performed by the developer while programming in the form of small tests ensuring that method behavior and output is as expected, these tests will not be documented.

At the end of each sprint cycle, we will perform integration testing of the portion of the application we have developed, using gray-box testing. This will ensure that the units of code added interact properly and that the classes function correctly.

At the end of the development process, when the application is finished, we will perform acceptance testing using system testing ourselves and user testing will be performed by users unfamiliar with the program using scenario testing, where these users will be given a list of tasks to perform by us and will not be given assistance when they perform them. These users also will be asked to give feedback about the quality of the application. All of this will be done using black-box testing.

2.5.4 Scope of tests

The scope of the integration tests will be the individual sub-portions of the program that are covered by the respective test cases.

The scope of the system tests and the scenario tests will be all the functional and non-functional requirements.

2.5.5 Test identifiers

Tests will be identified by a combination of letters specifying the testing method used to perform the test suffixed by one or two numbers, x and y. X specifies the sprint the test was performed in while y specifies the index of the test.

Integration tests	INTG-xy
System tests	SYST-y

Table 2.2: Test identifiers

2.5.6 Test priorities

Each test case will be given a priority of either low, medium or high. This priority describes the severity of a test failure. All system tests have a high priority, and the priority is therefore not specified in the individual system test cases. This section will define what each of these given priorities entails.

Low

A test case given that has been given a low priority is a case testing functionality of the application that exists as a conveniance to the user. The application is fully functional and usable even if this functionality is completely removed.

Medium

A test case that has been given a medium priority is a test case testing functionality that would signifcantly reduce the usability of the applicaton if it were removed, but the application would still work even without this functionality.

High

A test case that has been given a high priority is a test case testing functionality that is central to the application, and if this functionality were to be removed the application would no longer be usable.

2.5.7 Test case templates

Item	*Description*
Name	The name of the test case
Identifier	The identifier of the test following the convention defined in subsection 2.2
Testing technique	The testing technique used to perform the test
Features to be tested	A short description of the features to be tested.
Priority	The priority of the test, and severity of a test failure
Pre-conditions	A description of the pre-conditioned that need to be fullfilled before the test begins
Execution steps	A stepwise description of how the test is executed
Success criteria	The criteria that must be met for the test to be considered successfull
Test result	The result of the test
Test responsible	The person that executed the test

Table 2.3: A template for documenting integration tests.

Item	*Description*
Name	The name of the test case
Identifier	The identifier of the test following the convention defined in subsection 2.2
Testing technique	The testing technique used to perform the test
Requirements to be tested	A listing of the requirements this test case tests
Pre-conditions	A description of the pre-conditions that need to be fullfilled before the test begins
Execution steps	A stepwise description of the tasks to be executed
Success criteria	A listing of the criteria that must be met for the test to be considered successfull
Test result	The result of the test
Test responsible	The person that executed the test

Table 2.4: A template for documenting system tests.

2.6 Architecture plan

2.6.1 System outline

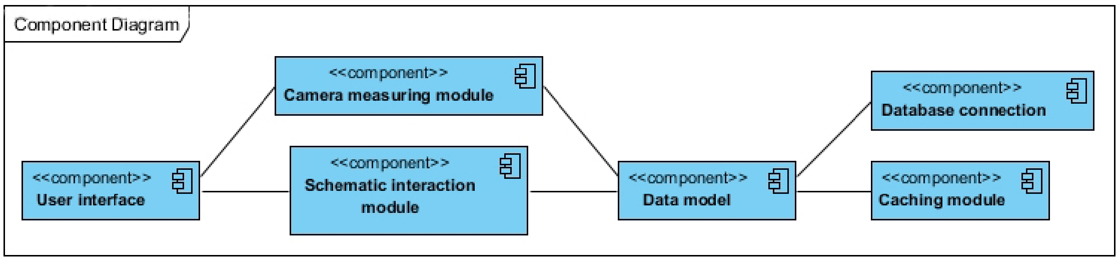

Figure 2.2: A diagram showing the main components of the system

The user interface A component that allows users to interact with the application. It consists of a large number of constructive elements provided by the iOS SDK, and fewer custom building blocks.

Schematic interaction A component that allows the user to create, view and interact with the plan. This is one of the main components of this system.

Caching A component that is used for caching files locally with a goal to reduce the data flow through wireless and mobile networks, because that can be slow and expensive.

Database connection A component that allows connecting to a database in a modular way, and adds flexibility to the design. It ensures that if the database type changes over time, the application logic can remain the same.

2.6.2 Class diagram

A diagram on how to structure the classes for the drawing screen. It is intended for planning purposes, and will be subject to change as the implementation progresses. The diagram was initially needed in order to visualize the class structure early in the development. This can avoid some major rewrites.

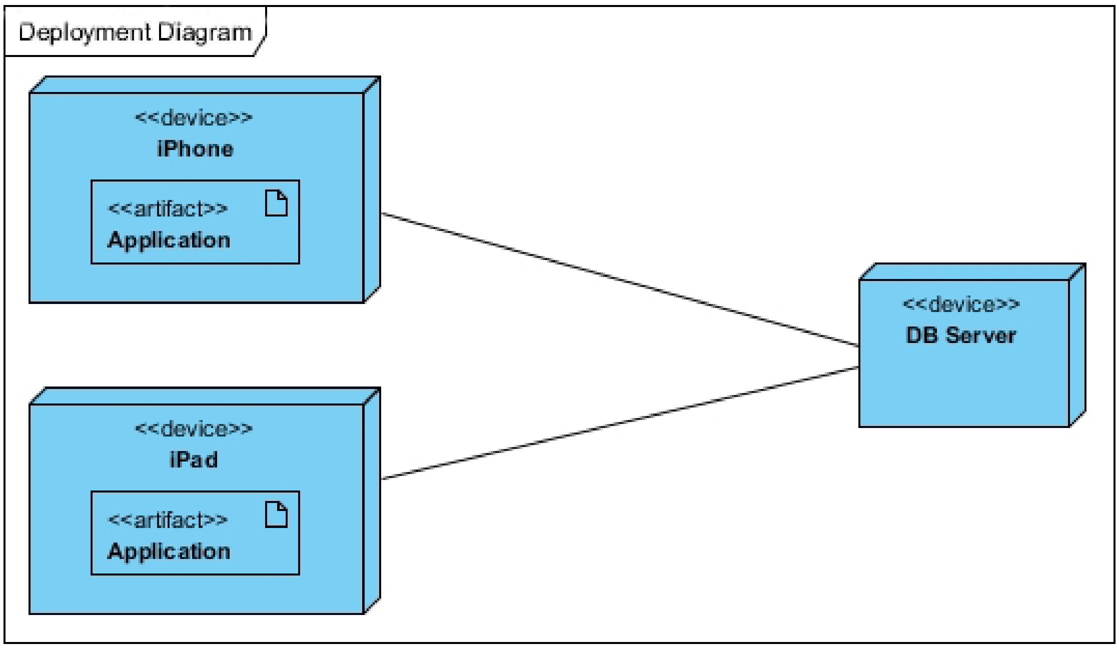

Figure 2.3: A diagram that illustrates what deployment platforms the app will utilize

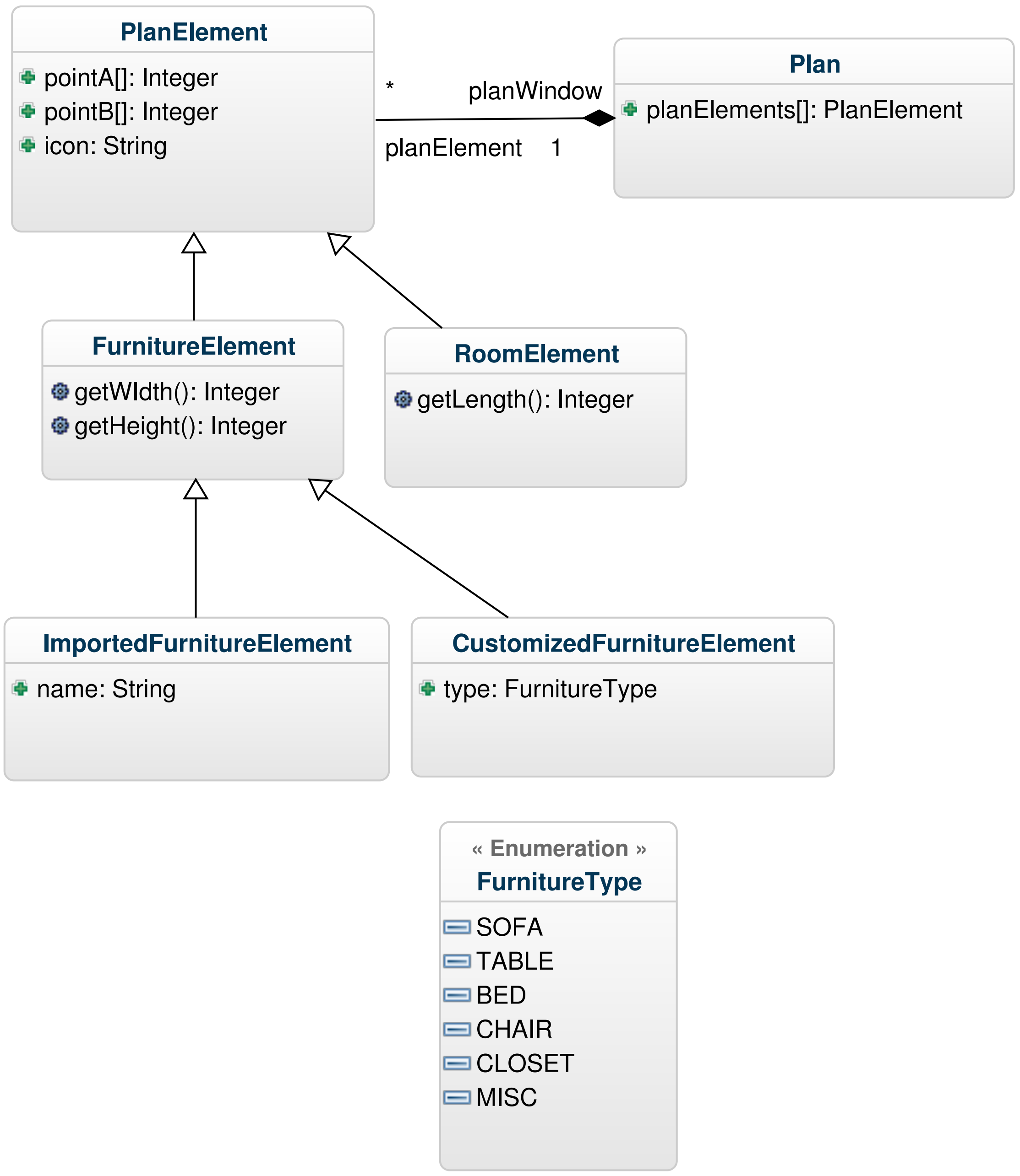

Figure 2.4: An initial class diagram

Chapter 3

Preliminary study

3.1 Development methodology

3.1.1 Scrum

Scrum is a methodology for project management, mainly used in the context of software development. It is an agile development, which means it follows the principles of the agile manifesto[7]. These include regular customer feedback, self-organizing teams, frequent deliveries and ease of adaptation to changes. Scrum is an iterative and incremental process which builds upon the idea of dynamically developing a product.
When following this developmental methodology, the overall project is divided into smaller segments. The customer specifies a set of requirements, and the development teams based on these create a product backlog. The product backlog is a master list of every desired feature the project shall have, where each item in the list also usually has an associated priority and workload estimate.These product backlog items are called stories. For each scrum iteration, calleda sprint, the development team selects one or several items from the productbacklog that they implement in that iteration. These stories are then translated into specific tasks, and this set of tasks is called the sprint backlog.
At the start of each sprint, a sprint preplanning meeting is held where the team decides which stories they want to work on in that sprint. This is typically done by selecting the stories with the highest priority in the product backlog.They then translates these stories to sprint backlog items and distribute the work throughout the team. During the sprint daily sprint meetings, called the daily scrum, are held where each development team member answers the following questions:

- What did I do yesterday?
- What will I do today?

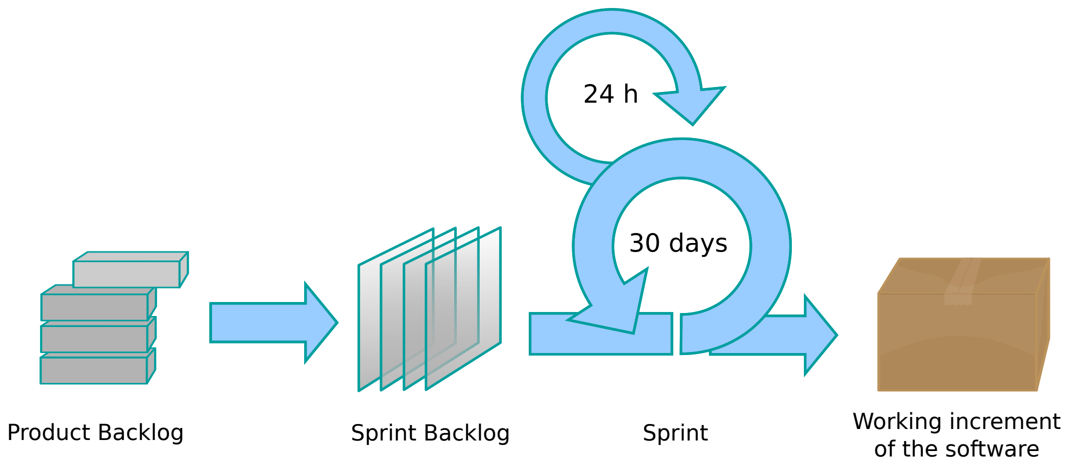

Figure 3.1: Figure depicting the Scrum process.

- Am i encountering any problems?

These daily meetings are a central concept in the scrum methodology, and are also where its name originates.

3.1.2 Waterfall

The waterfall model is a sequential software design process, in which the progress is seen as flowing steadily downwards through the different phases of the development process. The model is regarded as the first formalized approach to software development. Winston Royce, one of the first people to formally describe the model in the context of software development [9], defined the model to consist of the following steps:

- Requirements specification
- Design
- Implementation
- Testing
- Installation
- Maintenance

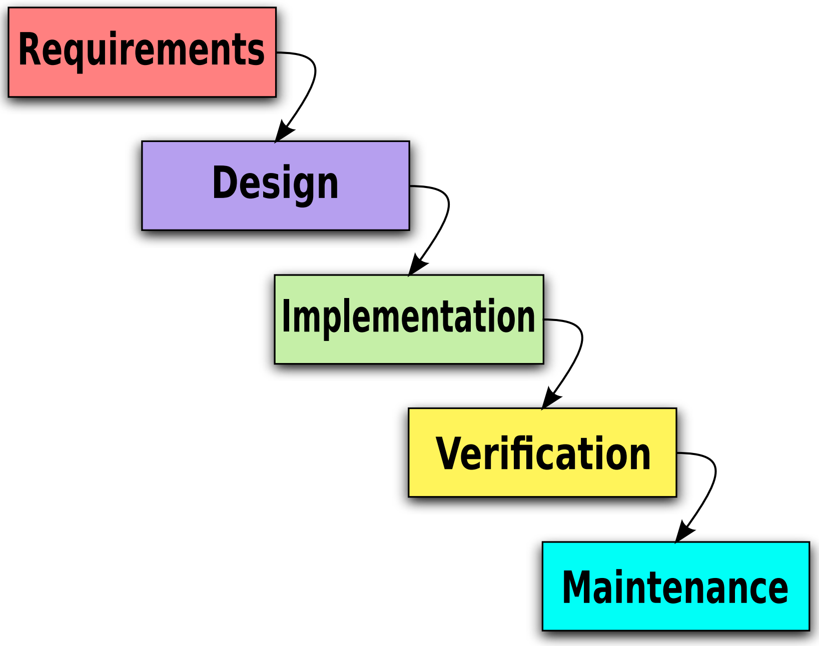

Figure 3.2: Figure depicting the waterfall model.

Each phase on the waterfall model is fully completed before moving to the next phase. Often, the testing and installation steps are merged into a verification steps.

3.1.3 Waterfall vs Scrum

Waterfall and Scrum both have their advantages and disadvantages when used in the context of software development.
The main supporting argument for using the waterfall model is that time spent early on in a project finding a realistic way to cover the requirements and creating a thorough plan for design and implementation will greatly decrease the amount of problems encountered later. Steve McConnell states the following in his article "Software Quality at Top Speed"[8]:

> Some project managers try to shorten their schedules by reducing the time spent on quality-assurance practices such as design and code reviews. Some shortchange the upstream activities of requirements analysis and design. Others–running late–try to make up time by compressing the testing schedule, which is vulnerable to reduction since it's the critical-path item at the end of the schedule.
>
> These are some of the worst decisions a person who wants to maximize

> development speed can make. In software, higher quality (in the form of lower defect rates) and reduced development time go hand in hand.

The problem with the Waterfall model is that very often, especially when the project owner is an external customer, it is unrealistic to expect that a clear, detailed and precise definition of the project objective exists. This is a problem that the flexibility and adaptiveness of Scrum solves, and is the main reason why Scrum has become the dominant methodology for software development in recent years.
The decision of which development methodology to choose then boils down to the following question: "Will I have a clear and correct picture of the complete project span at the beginning of the project?"

3.1.4 Our choice of development methodology

The customer for our project is the Norwegian governmental welfare agency, an organization that has very little knowledge in the field of software development and specifically application design for mobile technology. In our initial meeting with the customer, we were given a description of the desired functionality of the application to be created. During this meeting it was made clear to us that the representatives from the organization with whom we would be dealing with throughout the span of the project had a lot of experience with the tasks within the field of the intended usage of the application, but had very limited technical expertise.
In this meeting we also agreed that the final application functionality would, because of the limited time span and possible risks we could encounter, be subject to change and would be agreed during continous contact and feedback throughout the project.
Because of this it was apparent to us that Scrum was the correct methodology for this project due to it's adaptiveness and the fact that it is well-suited for iterative development, and this is the methodology we chose.

3.2 Technological aids

This section presents the tools and applications that were used during the project. Alternative tools are also briefly discussed, explaining why one tool were chosen instead of another.

3.2.1 Development technology

Xcode The Xcode integrated development environment (IDE) is the software development tool provided by Apple for OS X and iOS development. It is available as a free download from the Mac App Store for the newest OS X versions. The IDE contains all the tools required by developers, including a user interface builder and analysis of syntax and semantics.
Included in the IDE suite, there is a modified version of the GNU Compiler Collection, and the LLVM Compiler. These back-ends provide Xcode the support for C, C++, Objective-C, Objective-C++, Java, AppleScript, Python and Ruby. Additionally, the LLVM Compiler makes code suggestions and presents relevant documentation when needed. Starting with version 4.2, the default compiler is the Apple LLVM Compiler.
When creating applications for iOS, a simulator offers easy debugging and testing. It provides most of the functionality of a real iOS device, including simulation of the touch screen. This makes it trivial to test the application, especially the user interface. [2]
Xcode is the only real option when developing for iOS and other Apple technologies, which is the reason we are using it.

LaTeX LaTeXis a document preparation system based on the typesetting software TeX. It was originally developed in 1985 as a writing tool for mathematicians and computer scientists in order to typeset technical documents properly. The idea is that the author should not be too concerned about the layout of the document, and focus on the actual content instead. Later, LaTeXhas been developed by a growing community, and is now used for publishing almost every type of document.
For large documents like this project report, LaTeXmakes the organization easier because it automatically handles cross-referencing, tables and figures, page layout and bibliographies. Compared to WYSIWYG editors like Microsoft Word and Google Drive, a LaTeXdocument is plain text, and can be managed with Git like the rest of the project.
The learning curve for LaTeXcan be steep, but Johan is familiar with it, and recommended it for this project.

XML Extensible Markup Language (XML) is a markup language designed to store and transport data. In many ways, it resembles HTML, but the main difference is that HTML is for displaying information, whereas XML is for carrying information. Also different from HTML is that the tags in an XML document are not predefined, but are instead defined as the author inserts them. This makes the language a very flexible tool for storing any type of data. XML features a simple document structure in plain text, which makes it easily compatible with future

versions or other programs.
The decisive reason for choosing XML as the storage medium for plans was that we already had a finished XML parser at our disposal, tailored to Xcode (3.2.1). JSON is another similar format we considered, but based on the reason explained, we decided to choose XML.

3.2.2 Collaboration technology

Git[1] Git is a free and open source source code management system. Its focus is on speed, while featuring functions that makes it one of the most popular tools in software development [10]. In particular, strong support for non-linear development, distributed development, and efficient handling of large projects are core functionalities of Git. Linus Torvalds originally created Git for management of the source code for the Linux kernel [5].
While Git offers a lot more functionality than we require for this project, it can easily be used for simple revision tracking and collaboration after having learned the basics. Using GitHub in order to utilize the powerful tools for distributed development shipped with Git makes collaboration on code and the report easy and user friendly.
We also discussed the use of SVN because most of the group members had used it before, but the fact that Xcode (3.2.1) provides tight integration with Git, combined with Johans experience, made the group decide to use Git.

Google Drive[2] The amazing collaboration technology offered by Google Drive, allowing many people to work on the same documents at once, makes it an excellent choice when it comes to collaboration. It is an easy and user friendly alternative to other office suites like LibreOffice and Microsoft Office, with the big difference that it is web based and made with collaboration in mind.
In this project, we mainly use Google Drive for sharing useful information between the team members, and for writing parts of the report in an easy way before adding it to the LaTeXdocument.

Skype[3] Skype is an application for making video and voice calls over the internet. We use Skype sometimes for meetings

[1]http://git-scm.com/
[2]https://drive.google.com/
[3]http://www.skype.com/

with the supervisor, and when we have small group meetings and don't need to meet in person.

Chapter 4

Quality assurance

In this chapter we will describe the guidelines we have created to ensure efficient communication within the group, and between the group and the customer and supervisor.

4.1 Group interaction

4.1.1 Group communication

Group communication shall take place using e-mail. A google group has been created, which shall be used by members of the group to send e-mails to the rest of the group.

4.1.2 Group meetings

At the start of each sprint, the group shall meet to discuss the plan during the sprint.
At the end of each sprint, the group shall meet to review and evaluate the sprint. There shall be three scrum meetings each week to keep everyone up to speed during the scrum sprints. If one or several group members are unable to attend in person, these meetings will be held over skype. These meetings will be held on Mondays, Wednesdays and Fridays. If a group member is unable to participate in the group meeting, the rest of the group shall be notified via e-mail as soon as possible.

4.1.3 Group document sharing

Documents and files shall be shared using google drive.

4.2 Supervisor interaction

The group shall meet with the supervisor at least once per sprint to review progress and discuss further work. The point of emphasis in these meetings will be to discuss progress on the report and if the customer interaction is functioning properly. If this meeting can not be held in person it shall be held over skype. Time and place for the next meeting should be decided at the conclusion of each meeting. Minutes from these meetings shall be written by the secretary and shared with the group using git.

4.3 Customer interaction

The group shall meet with the customer at the start of each sprint and at the end of each sprint. When in the transition between two sprints, these meetings can be combined so that the end-of-sprint meeting and the pre-sprint meeting with the customer are the same. The point of emphasis in these meetings will be to discuss the progress of the application. Time and place for the next meeting should be decided at the conclusion of each meeting. Minutes from these meetings shall be written by the secretary and shared with the group using git.

Chapter 5

Requirements

5.1 Functional requirements

Following is a list of requirements, each assigned a unique ID in order to refer to them later.

FR1 Create new plan
User can create a new plan as an XML document with a name of their chosing

FR2 Open plan
User can open an existing plan from an XML document for preview or modification

FR3 Save plan
User can save plan as an XML document

FR4 Add walls and other elements
User can add walls and other custom elements to a plan

FR5 Add custom elements
User can add a custom element to a plan, adjusting its dimensions as desired

FR6 Add database elements
User can add predefined elements from NAV's database to a plan

FR7 Move elements
User can move elements within a plan

FR8 Resize elements
User can resize elements within a plan

FR9 Rotate elements
User can rotate elements within a plan

FR10 Remove elements
User can remove elements from a plan

FR11 Display dimensions
User can view real dimensions of plans and elements

FR12 Detect collision between elements
User can if different elements collide while moving them within a plan

FR13 Camera measurement
User can define the walls of a room using measurements from a camera

FR14 Zoom
User can zoom a plan in and out

FR15 Pan
User can navigate around in a plan.

FR16 Export
User can export the plan to other mobile devices for review and editing.

5.2 Non-functional requirements

NFR1 Usability

Since the application is designed for a wide range of users, it needs to be as simple as possible to use. The user interface should be simple and intuitive because most people in the target group has no experience in using similar applications.

NFR2 Maintainability

This project is intended to be developed further after this course, so the application should be designed with modularity in mind, in order to make it easier to expand and upgrade later. Maintaining existing systems and keeping consistency is one of the primary goals.

NFR3 Performance

It's very important to have a good response because of the overall user experience and the measurement accuracy. Every action must be completed in less than 0.1 second. Transitions between screens must not be longer than 0.5 seconds.

NFR4 Efficiency

Because the application is intended for use on a daily basis in a business environment it is important that it is efficient to use as the point of the application is to streamline the work of the employees who will be using the application. The application is intended to replace using sheets of paper, but if it is slow and clunky to use then paper drawings will be preferable.

5.3 Use cases

Figure 5.1 illustrates an abstraction of the requirements specified by the customer. It is intended to give an idea of the relations between the main requirements of the application. For example, if you choose to load a plan, you will be directed to the plan management where you will be able to make changes to the plan.
For more detailed use case descriptions, see table 5.1.

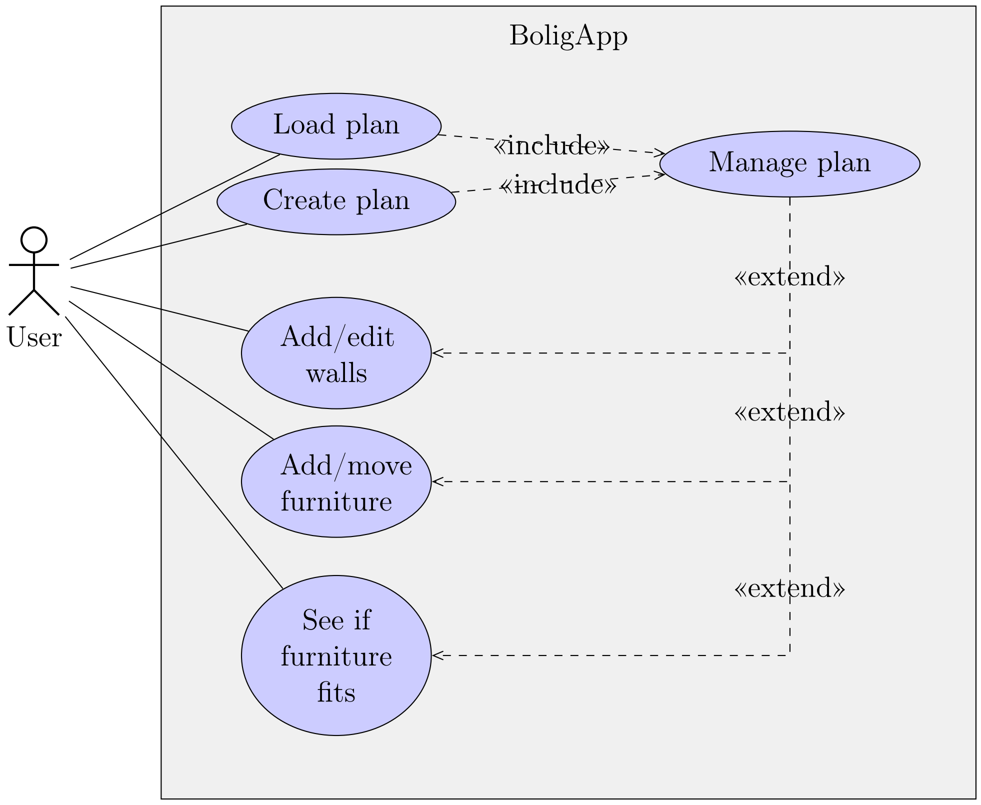

Figure 5.1: Use case diagram

Table 5.1: Table of use cases

Start conditions	*Steps*
Create new plan (FR1)	
• User has opened the application and is on the main screen	1. User presses “Ny plan” 2. User fills in data about the house 3. User presses “Lag plan”
Load plan (FR2)	
• User has opened the application and is on the main screen • A plan is already created and available	1. User presses “Åpne plan” 2. User selects the desired plan from the list 3. User presses “Åpne”
Draw room (FR4)	
• A plan is created (see *Create new plan*) • The application is in the plan view	1. User selects "Vegg" from the list of items 2. User draws the wall 3. Repeat 2. for as many corners as the user desires
Add a piece of furniture (FR4)	
• A plan is created (see *Create new plan*) • A room is drawn (see *Draw room manually*) • The application is in the plan view	1. User presses “Legg til” 2. User selects a category from the list that appears 3. User selects the item to add 4. User drags the newly created item to the desired location, and specifies the orientation
Export plan (FR16)	

Continued on the next page

Start conditions	*Steps*
• A plan is created (see *Create new plan*) • The application is in the plan view	1. User presses “Eksporter” 2. User selects the file format to export as from a list 3. User presses “Ok”

Part II

Sprints

Chapter 6

Sprint 1

6.1 Duration

The duration of sprint one was from the 2nd of September to the 13th of September.

6.2 Scrum master

The scrum master for sprint one was Johan Reitan.

6.3 Backlog

ID	*Task*	*Estimated*	*Actual*
1	**Create UI**	**27**	**29**
1.a	Create views	2	4
1.b	Create view elements	10	10
1.c	Design UI	5	5
1.d	Learn XCode components	10	10
2	**Implement XML support for holding plans**	**8**	**19**
2.a	Design XML structure	4	4
2.b	Create XML parser	2	6
2.c	Create a few XML samples	1	4
2.d	Testing	1	5
3	**Display a plan loaded from XML**	**45**	**36**
3.a	Load walls	10	7
3.b	Load elements	10	6
3.c	Load wall sizes	2	2
3.d	Testing	8	10
3.e	Learning `QuartzCore`	15	11
4	**Write documentation**	**10**	**10**
Total hours		*90*	*94*

Table 6.1: Backlog for sprint 1

ID	*Task*	*Estimated*	*Actual*
1	Create UI	27	29
2	Implement XML support for holding plans	8	19
3	Display a plan loaded from XML	45	36
4	Write documentation	10	10
5	Sprint planning	2	2
6	Sprint meetings	3	3
7	Supervisor meeting	6	6
8	Customer meetings	12	12
9	Lectures	22	22
10	Write report	65	46
Total hours		*200*	*185*

Table 6.2: Total work hours for sprint 1

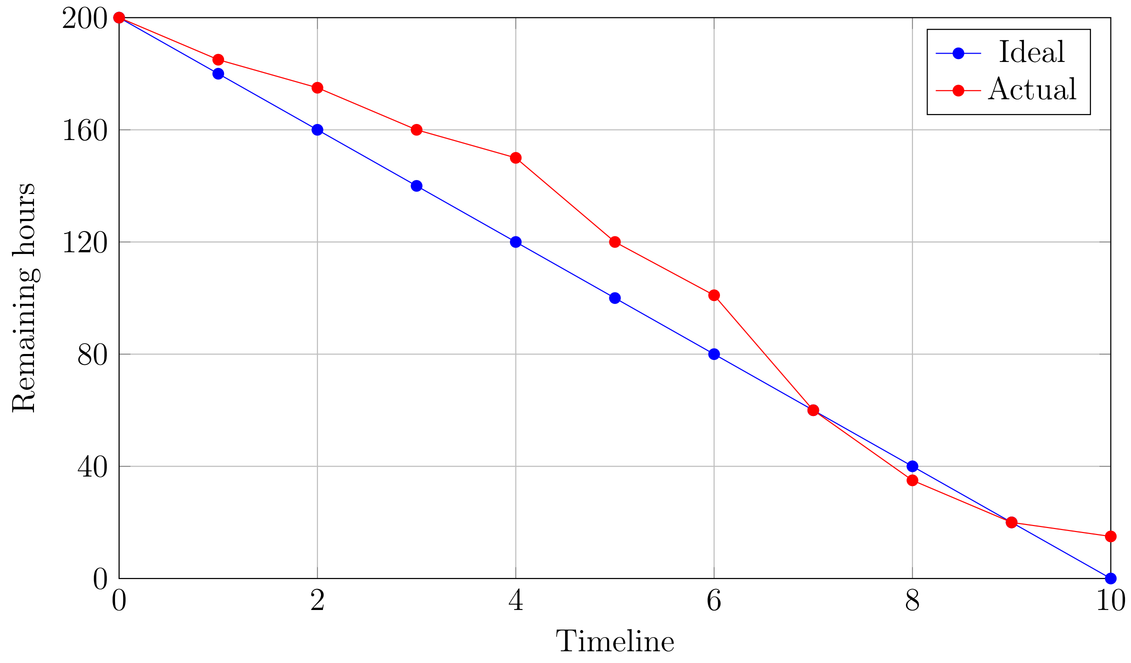

Figure 6.1: Burn down chart for sprint 1

6.4 Goals

The main goal of sprint one was to create a user interface as we wanted to have something to show the customer for the next agreed meeting. In agreement with the customer, we decided that the first prototype needed to be able to show elements from an already defined plan. To accomplish that, a lot of back-end stuff needed to be done, so we decided that the first prototype would not be very user interactive.

6.5 Design and implementation

In this section we will describe the design and implementation of the XML parser, the user interface, and the functionality for drawing elements.

6.5.1 User interface design

For designing and building the user interface we have used the interface builder included in Xcode (more on Xcode: 3.2.1). The main menu and all of the other views are connected using storyboards and navigation controllers. Some parts of the user interface in the room plan will have to be implemented progammatically, and will be included in the later sprints.

6.5.2 XML Parser

According to requirement **FR2**, the system needs to be able to load a plan (see section 3.2.1 for more about XML).
For the purpose of parsing data from an XML file, we have declared and implemented the following 3 functions:

```
- (void)loadWalls:(TBXMLElement *)element toPlan:(NAVPlan *)plan;
```

This function doesn't return anything, because its purpose is to create plan elements of type `NAVWall` and put them into the array of plan elements. The parameter `element` is used as a root element in the XML file, and the `plan` parameter is the plan in which elements are going to be saved.

```
- (void)loadRectangularElements:(TBXMLElement *)element toPlan:(
      NAVPlan *)plan;
```

This function also doesn't return anything, because its purpose is to create plan elements of type `NAVRectangularElement` and put them into the array of plan elements. Since `NAVRectangularElement` and `NAVWall` have different attributes, we had to create 2 separate functions for parsing. The parameter `element` is used as a root element in the XML file, and `plan` parameter is the plan in which elements are going to be saved.

```
- (NAVPlan *)loadPlanFromXML:(NSString *)URL;
```

As one can understand from its name, this function is used to load a plan from an XML file. Functions `loadWalls` and `loadRectangularElements` are both called inside this function. The `URL` parameter specifies the location of the XML file.

6.5.3 Drawing elements

A part of the core functionality of our application is the ability to draw plan elements on the screen (walls, doors, furniture etc.). The `QuartzCore` [3] framework provides useful functions for transforming (moving, resizing, rotating etc.) images. We decided to use `QuartzCore`, and to represent elements as the `UIImageView` type, because its function `(void)drawElements` contains all of the code required for drawing a plan on the screen. That function is called in the `(void)viewDidLoad` method located in our `NAVPlanViewController` class.

6.6 Testing

The focus area of this sprint was mainly implementing "back-end" functionality such as XML file structure. This meant that the amount of testable program functionality produced would be quite small compared to work hours spent. Therefore, the test set for this sprint only consisted of three cases. All of these items tested were absolutely essential to the functionality of the overall application, and were given a high priority.

6.6.1 Test cases

After the implementation phase of this sprint was completed, three tests were performed:

1. Navigating between windows: *Test case INTG-11*
2. Opening a created plan: *Test case INTG-12*
3. Adding an item to the plan: *Test case INTG-13*

See appendix C.1 for a detailed summary of the test executions and results.

6.6.2 Test evaluation

All tests in this sprint were successfull. This was no surprise, as a lot of the functionality implemented in this sprint was not very complex.

6.7 Deliverables

At the end of this sprint, we were able to finish and present to the customer the following requirements:

- **FR2**: *Open plan*

6.8 Customer feedback

At the customer meeting, we presented to the customer the main menu and how to navigate through the program, in addition to how to load a plan and an initial draft at the plan-viewer component of the application. The customer understood that the focus area of this sprint had mainly been implementing back-end components of the application, but was pleased that we had prioritized usability of the application in the menu navigation by making the buttons large and the labels easy to read.

6.9 Sprint evaluation

This section will take a look at things that we have accomplished and include a quick summary of what went well and what didn't go so well during this sprint.

6.9.1 Positives

All of the implementation goals of the first sprint were fulfilled. An intuitive, albeit very simple user interface was created, and we showed the customers how to load a plan in our customer meeting. The customer was very satisfied by our presentation of application. We also managed to reach first milestone of our plan was to load and display display plan.

6.9.2 Negatives

Unfortunately, the documentation part of the sprint wasn't handled very efficiently. We had some problems with structuring the document, and in general we focused much more on creating working prototype then writing proper documentation.The supervisor suggested that we should make our report more formal, and in general he had a lot of suggestion about our document subsections and their order. The sprint planning was also quite poor, which we noticed when we needed to do a lot of refactoring in sprint 2.

Chapter 7

Sprint 2

7.1 Duration

The duration of sprint two was from the 16th of September to the 27th of September.

7.2 Scrum master

The scrum master for sprint two was Dusan Stefanovic.

7.3 Backlog

ID	*Task*	*Estimated*	*Actual*
1	**Refactor code**	**5**	**16**
2	**Create interactive plan**	**60**	**58**
2.a	Functionality to move elements around in plan	15	17
2.b	Check for intersection between two elements	15	11
2.b.i	Separating Axis Algorithm	5	3
2.c	Check for intersection between wall and element	5	6
2.d	Functionality for drawing labels	5	4
2.d.i	Point in polygon function	2	2
2.e	Testing	10	12
2.f	Learn about `TouchBegan`, `TouchMoves` and `TouchEnds`	10	8
3	**User Interface**	**15**	**15**
3.a	Create resize view	15	15
3.a.i	Adjust text fields	2	1
4	**Write documentation**	**10**	**12**
Total hours		*90*	*101*

Table 7.1: Backlog for sprint 2

ID	*Task*	*Estimated*	*Actual*
1	Refactor code	5	16
2	Create interactive plan	60	58
3	User interface	15	15
4	Write documentation	10	12
5	Sprint planning	2	2
6	Sprint meetings	3	3
7	Supervisor meetings	12	12
8	Customer meeting	6	6
9	Write report	87	69
Total hours		*200*	*193*

Table 7.2: Total work hours for sprint 2

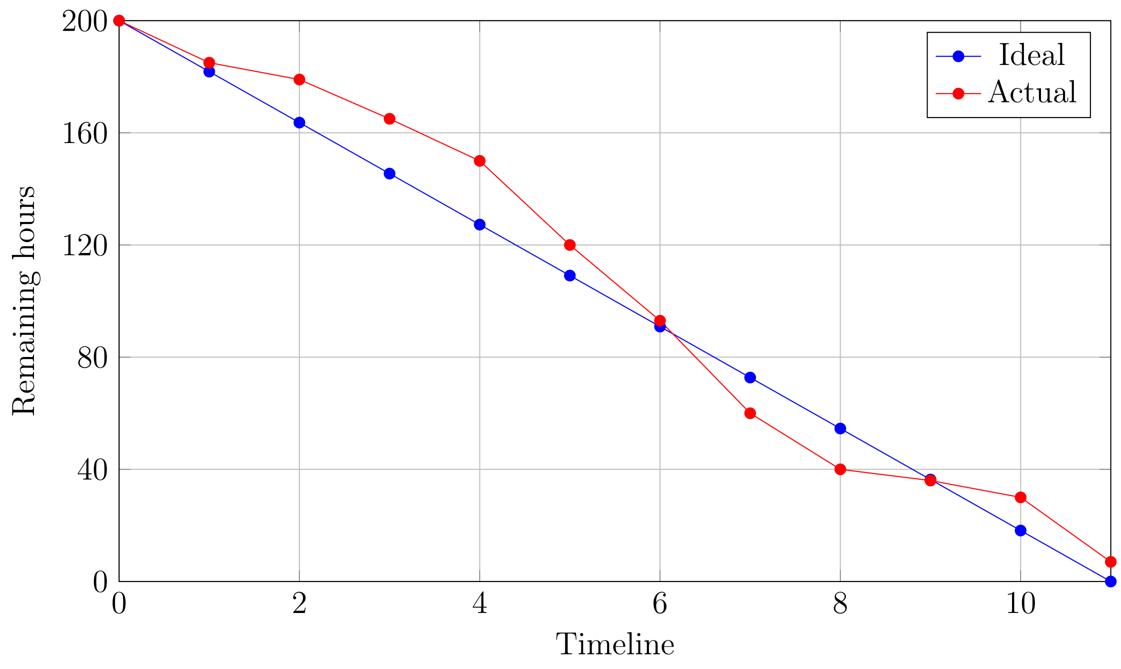

Figure 7.1: Burn down chart for sprint 2

7.4 Goals

Making the plan interactive was the main goal of sprint two. The ability to move elements around as well as resizing, rotating and removing them was implemented. In addition the functionality to notify the user if one element intersects with another during translation or rotation was added. All of this was done using predefined elements; the ability to draw new plan elements was not yet implemented in this sprint. Another goal for this plan was code refactoring: we wanted to ensure that our code was clean and easy to read for further development.

7.5 Design and implementation

7.5.1 Interactive plan

According to requirement **FR12**, the application should be able to detect if the user wants to move an element to a position where it collides with another element. The `-(void)drawElements` function was refactored and now it consists of two functions containing code for drawing the Wall or Rectangular elements:

```
(UIImageView* )drawElementAtLocation:(CGPoint)location eName:(
      NSString *)name eIcon:(NSString *)icon eWidth:(CGFloat)width
      eHeight:(CGFloat)height eAngle:(CGFloat)angle;
```

```
-(void)drawWallElement:(NAVWall *)element
      FromPlanElementwithStartPoint:(CGPoint)startPoint endPoint:(
      CGPoint)endPoint doorsAndWindows:(NSMutableArray *)
      elementDoorsAndWindows;
```

These functions draw an element at an exact place in the plan according to attributes read from XML. Note that these functions only draw elements, but do not add them to the plan, as this will be implemented in sprint 3.

```
-(CAShapeLayer *)drawWallElementLayerwithWallLayer:(CAShapeLayer *)
      elementLayer startPoint:(CGPoint)startPoint endPoint:(CGPoint
      )endPoint;
```

This is a helper function that draws a wall on an element. Because its return type is `CAShapeLayer`, it can be reused whenever a wall has to be drawn.

```
-(BOOL)view:(UIView *)view1 intersectsWith:(UIView *)view2;
```

This function checks if two elements are overlapping. Because elements can be rotated, the standard function `CGIntersectRect` cannot be used, so we implemented the Separating Axis Algorithm in order to check for overlap. SAT (Separating Axis Theorem) is based on the principle that “If two convex objects are not penetrating, there exists an axis for which the projection of the objects will not overlap” [4]. Also part of the algorithm are the functions:

```
-(BOOL)convexPolygon:(CGPoint *)poly1 count:(int)count1
      intersectsWith:(CGPoint *)poly2 count:(int)count2;
-(void)projectionOfPolygon:(CGPoint *)poly count:(int)count onto:(
      CGPoint)perp min:(CGFloat *)minp max:(CGFloat *)maxp;
```

```
-(void)drawLabel:(id)element fromStartPoint:(CGPoint)startPoint
      toEndPoint:(CGPoint)endPoint withPoints:(CGPoint *)pointss
      pointsCount:(int)pCount;
```

This function is used to draw labels containing the sizes of walls. The size should always be written on the outside of the room, which requires some logics. The argument `(CGPoint *)pointss` is an array of all wall corners in the plan, and `(int)pCount` is the number of corners. After the centre of the wall line is calculated from the points `CGPoint)startPoint` and `CGPoint )endPoint`, two points are created: one on opposite sides of the line center. Then, the function `bool pointInPolygon(int numOfPoints, CGPoint *points , CGPoint point)` is called to check whether any of the newly created points are inside the polygon made from the vertices of the wall, and one on the outside is chosen as the label's central point.

```
-(BOOL) intersectionOfLineFrom:(CGPoint)p1 to:(CGPoint)p2 ImageView
      :(UIImageView*)elementView;
```

The purpose of this function is to check whether there is an intersection between a rectangular element and a wall. The way it works is that a rectangular element is divided into the 4 line segments that is its edges. If the element is rotated, it is important to note that the end points of each line segment will have to be translated to that angle, and the line segments recreated. The helper function `-(BOOL)intersectionOfLineFrom:(CGPoint)p1 to:(CGPoint) p2 withLineFrom:(CGPoint)p3 to:(CGPoint)p4` checks if there is an intersection between two line segments.

```
-(void)UpdateElementViewWithUIimage:(UIImageView*)elementView;
```

This function is called when changes are made to any kind of element. Its purpose is to save changes to the current plan.

```
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
```

Handles the event when one of the elements is clicked. It saves the selected element, its centre and coordinates for where the touch was made.

```
-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
```

Handles events occurring when an element is dragged across the screen. The function for checking if there is an intersection between elements is called. If there is a conflict, the selected element turns red, and when the touch is released and there is still an intersection, the element is moved back to its original position.
If the new position of the element is valid, a dialog will open allowing the user to resize, rotate and move the selected element. The text fields on the dialog will only accept numbers and punctuation. The dialog can be moved across the screen. If an empty space is clicked, this dialog disappears.

Every wall has defined start and end points, which are used to move them around the plan. Elements only need to be touched in order to drag them to the desired position.

```
-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
```

Called when a touch is released. If the new position of an element is invalid, it will be set back to its original position.

```
-(void)btnResizeClicked:(UIButton*)button;
```

Called when the resize button on the resize dialog is pressed. It will apply the desired transformation to the selected element and update the resize dialog with new measurements. For modal dialogs we have used the `CODialog` library[1], adjusted to our needs.

```
-(void)btnDeleteClicked:(UIButton*)button;
```

Called when the delete button on the resize dialog is pressed. It removes the drawing of the element and the element itself from the plan..

7.6 Testing

This section introduced all of the main functionality for executing actions on plan-elements. The scope of the functionality implemented was clearly defined and it was easy to create a test set that covered this. The tests testing functionality to execute actions on plan-elements were all given a high priority, as they were central to the application being usable, while tests testing for intersecting lines were given a medium priority because this exists for the conveniance of the user, but the application would still be usable without it as the intersection would be visible.

7.6.1 Test cases

After the implementation phase of this sprint was completed, seven tests were performed:

- Moving an element in the plan: *Test case INTG-21*
- Checking for intersecting points between two items: *Test case INTG-22*
- Checking for intersecting points between wall and item: *Test case INTG-23*

[1] `https://github.com/eaigner/CODialog`

- Resizing an item: *Test case INTG-24*
- Rotating an item: *Test case INTG-25*
- Removing an element: *Test case INTG-26*

See appendix C.2 for a detailed summary of the test executions and results.

7.6.2 Test evaluation

All of tests were successfull except the test case testing for detection of intersecting lines between walls and items. Instead of snapping the item back to the original position, the item would be moved to an incorrect position. This problem was communicated to the person responsible for implementing this functionality and quickly solved.

7.7 Deliverables

At the end of this sprint, we were able to finish and present to the customer the following requirements:

- **FR7**: *Move elements*
- **FR8**: *Resize elements*
- **FR9**: *Rotate elements*
- **FR10**: *Remove elements*
- **FR12**: *Detect collision between elements*

7.8 Customer feedback

The initial reaction of the customer to the functionality added to the application in this sprint was positive, and they were happy that we had understood the concept of what they wanted developed. They were happy with the wave we had implemented performing different actions on elements in the plan, and that we had covered the requirement of checking for intersecting points between plan elements.

7.9 Sprint evaluation

This section will take a look at things that we have accomplished and include a quick summary of what went well and what didn't go so well during this sprint.

7.9.1 Positives

Like in sprint one, we were satisfied with how the coding part of this sprint went. We managed to fulfill all of the customer requirements (see 7.7) that we hoped, and they were pleased with the application during our customer meeting at the end of the sprint. All of the high priority tests testing functionality of the actions implemented on the plan elements were successful.

7.9.2 Negatives

Because of some sloppy implementation in sprint one, we had to recode or refactor a big portion of the code. This refactoring took some time to finish. Documentation still proved to be problem for us, as we finished most of the documentation for sprint one during sprint two, we didn't have time to finish documentation for sprint two on time. Because of this, we decided to pause developing the application while we caught up with the documentation at the beginning of sprint three. We also had some illness to one of our team member in this sprint, resulting in one of the team members missing two scrum meetings and slowing down both the work on the application and documentation.

Chapter 8

Sprint 3

8.1 Duration

The duration of sprint three was from the 30th of September until the 11th of October.

8.2 Scrum master

The scrum master for sprint three was Nenad Milosavljevic.

8.3 Goals

The main planned goal of sprint three was the ability to add new elements to the plan. The ability to zoom and pan around in the plan viewer was also a goal of this sprint. The customer specified in the sprint pre-planning meeting that they wanted elements like doors and windows to be docked to walls. In addition they wanted us to make a few altercations to the user interface. We also decided that creating new plans, in addition to saving and loading plans was to be implemented in this sprint, since previously we had just been working on a pre-defined sample plan.

8.4 Backlog

ID	*Task*	*Estimated*	*Actual*
1	**Refactor code**	**5**	**5**
2	**Create interactive plan**	**60**	**65**
2.a	Add doors and windows	5	4
2.b	Draw new elements	5	6
2.c	Draw new walls	5	3
2.d	Add new elements to plan	5	5
2.e	Add new walls to plan	5	7
2.f	Add gesture recognizers	5	10
2.g	Create menu for new items	10	10
2.g.i	Get to know the library used	6	5
2.g.ii	Add elements to list dynamically	2	1
2.g.iii	Menu design	2	4
2.h	Testing	12	12
2.i	Learn about gesture recognizers	8	8
3	**User interface**	**5**	**1**
3.a	Create buttons on plan view	1	1
4	**Load plan dynamically**	**10**	**16**
5	**Create new plan**	**10**	**7**
6	**Write documentation**	**10**	**13**
Total hours		*100*	*107*

Table 8.1: Backlog for sprint 3

ID	*Task*	*Estimated*	*Actual*
1	Refactor code	5	5
2	Create interactive plan	60	65
3	User interface	5	1
4	Load plan dynamically	10	16
5	Create new plan	10	7
6	Write documentation	10	12
7	Sprint planning	2	2
8	Sprint meetings	3	3
9	Supervisor meetings	6	6
10	Customer meeting	6	6
11	Write report	83	75
Total hours		*200*	*198*

Table 8.2: Total work hours for sprint 3

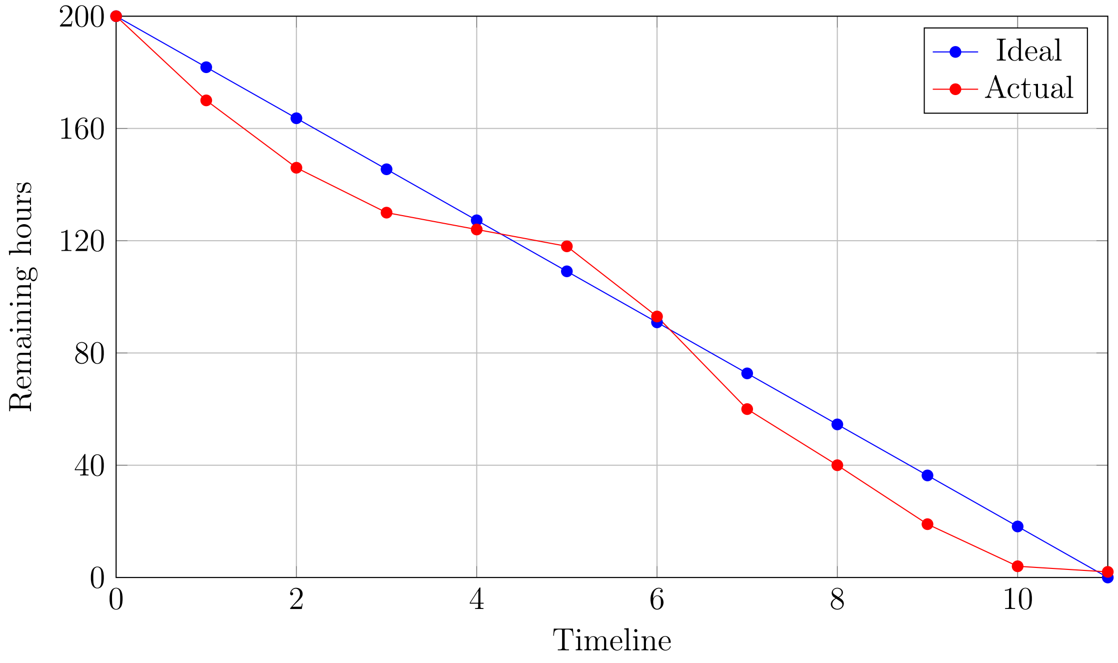

Figure 8.1: Burn down chart for sprint 3

8.5 Design and implementation

8.5.1 XML parser

For the purpose of saving a plan to an XML file, the `NAVParser` class was expanded with the `-(void)savePlan:(NAVPlan *)plan toXML:(NSString *) fileName` function. Plan elements will be saved to a named plan using the same parsing conventions as are used for reading a plan.

8.5.2 Interactive plan

For the creation of a slide menu in which the available items are located, we have used the `SWRevealViewControler` library [1]. We have changed some code to suit our needs, as we need items to be loaded dynamically.
We have added a gesture recognizer for scaling and rotating elements that makes the user able to transform elements not only through the resizeview but also using touch gestures on screen. For this purpose, we have added `UIPinchGestureRecogniser` and `UIRotateGestureRecogniser` which are instanced in the `viewdidload` method.

```
-(void) addElementToPlanElementName:(NSString *)name usingImageView
      :(UIImageView *)imgView;
```

This function is used after a new element is drawn on a plan. Its purpose is to add a new element to the array of plan elements.

```
-(void) addWallElementToPlanwithStartPoint:(CGPoint)startPoint
      endPoint:(CGPoint)endPoint;
```

This function is used after a new wall is drawn on a plan. The purpose is to add new wall to the array of plan elements. Drawing a wall is done in the `touchesMoved` function by the already defined (in section 7.5.1) function:

```
-(CAShapeLayer *)drawWallElementLayerwithWallLayer:(CAShapeLayer *)
      elementLayer startPoint:(CGPoint)startPoint endPoint:(CGPoint
      )endPoint;
```

[1] `https://github.com/John-Lluch/SWRevealViewController`

The `TouchesMoved` function now contains the part of the code that takes care of moving doors and windows around the plan. Doors and windows can only be placed on the wall, and whenever any of those elements changes, its wall function for update is called. As an array of doors and windows is an attribute of the `NAVWall` class, the parent of the element should be updated every time the element changes.

```
-(void)UpdateDoorOrWindow:(UIImageView *)doorOrWindow;
```

This function is called when there are some changes to a door or window element.

8.5.3 User interface

An action bar was added to the plan view, containing buttons for adding items, showing or hiding wall points and coordinates, and save and export buttons. The button functions are self explanatory and do not need any special explanation.

8.5.4 Loading plans dynamically

Instead of showing a static list of plans, the load plan view now dynamically presents all the plans from the specified directory.

8.5.5 New plan

The screen for creating a new plan is now working, and after the name of the plan is entered, it will show an empty plan with the ability to add items and save the plan.

8.6 Testing

This sprint introduced adding elements to the plan, and creating new plans. The important thing when testing this was to try and spot potential problems in the underlying xml files, because up until this point we had not actually been manipulating the underyling xml in a very complex manner. All of functionality involving adding adding elements, saving/loading plans and resizing and panning the plan viewer were given a high priority because they were all deemed central to the application. Correctly storing plan metadata was given a low priority.

8.6.1 Test cases

After the implementation phase of this sprint was completed, eight tests were performed:

- Navigating menu for adding items: *A template for documenting system tests.*
- Adding walls: *Test case INTG-32*
- Adding doors: *Test case INTG-33*
- Adding items: *Test case INTG-34*
- Saving and loading a plan: *Test case INTG-35*
- Plan metadata properly saved: *Test case INTG-36*
- Zooming the plan: *Test case INTG-37*
- Panning the plan: *Test case INTG-38*

See appendix C.3 for a detailed summary of the test executions and results.

8.6.2 Test evaluation

All tests in this phase were successfull.

8.7 Deliverables

At the end of this sprint, we were able to finish and present to the customer the following requirements:

- **FR1**: *Create new plan*
- **FR3**: *Save plan*
- **FR4**: *Add walls and other elements*
- **FR11**: *Display dimensions*
- **FR14**: *Zoom*

- **FR15**: *Pan*
- **FR16**: *Export*

8.8 Customer feedback

At the end of this sprint, the application had almost all of its core functionality implemented and the only large remaining part of the application to be implemented was the connection to the customer's database of assistive technology. We were able to let the customer test the application during the meeting at the conclusion of this sprint and receive feedback about what they though about the usability of the application. The customer performed a few typical tasks, while talking us through how they experienced executing these tasks.
The customer was pleased with the look of the application, and happy that we had got as far as we had done in the development process.
A recurring problem was that the freedom the application was giving the customer through allowing the customer to manually drag walls wherever he desired and rotate and resize elements through dragging them came with a drawback. Because of all the freedom we were giving the customer it was making typical tasks like creating a square room unnecessarily complex. In addition, the customer was repeatedly accidently panning and zooming in the plan when trying to perform actions on elements. Additionally, the customer found it difficult to accurately set element sizes, like wall lengths, through scaling the elements resizing in the plan. We agreed we would take measures to improve these areas in sprint four.

8.9 Sprint evaluation

This section will take a look at things we have accomplished and include a quick summary of what went well and what didn't go so well during this sprint.

8.9.1 Positives

Sprint 3 was the most productive sprint from the beginning to end, both in the way of coding and writing documentation. In this sprint, we were able to catch up in terms of working on our project report in unison with performing work on the application. We were very pleased to be able to implement what we hoped for in time for the customer meeting because this gave us a chance to receive a lot of valuable feedback in the customer meeting due to us being able to perform testing of the application prototype with the customer during the meeting.

8.9.2 Negatives

We had a technical problem with the computer that we planned to use for simulation of the application in the meeting with the customer at the conclusion of this sprint, resulting in the customer needing to wait for around 20 minutes until we managed to get the problem fixed.

Chapter 9

Sprint 4

9.1 Duration

The duration of sprint four was from the 14th of October until the 26th of October.

9.2 Scrum master

The scrum master for sprint four was Jørgen Faret.

9.3 Goals

The main goal of this sprint was to connect the NAV database for accessibility aids with our application. Unfortunately, because of some internal communication problems within NAV, we were unable to get access to the database. As a remedy for this problem, and to make the application as usable as possible we decided to focus on the creation of custom elements in this sprint. In addition to this, we also used this sprint to implement changes to the application that the customer had suggested after the demonstration of the functionality implemented in the previous sprint. These changes included:

- Being able to lock the plan viewer to avoid accidentally zooming and panning when trying to move elements.
- Zooming in the plan using buttons
- Copying and pasting elements (like walls)
- Creating horizontal and diagonal walls by entering the length of the wall

- Creating rectangular rooms by specifying room dimensions
- Removing the decimal points from element dimensions, because this degree of accuracy was unnecessary

9.4 Backlog

ID	*Task*	*Estimated*	*Actual*
1	**Refactor code**	**5**	**5**
2	**Delete a plan from the device (and the list)**	**4**	**5**
3	**Export a plan to PDF, PNG or native plan format**	**8**	**11**
3.a	Create dialog for exporting		2
3.b	Function for exporting to PDF		3
3.c	Function for exporting to PNG		3
3.d	Function for exporting to .nav		3
4	**Share plans over e-mail**	**10**	**9**
4.a	Create dialog for sharing .nav, PNG or PDF		3
5	**Import plan for editing**	**4**	**4**
6	**Draw a room element with predefined dimensions**	**2**	**3**
7	**Lock the plan zoom and pan**	**2**	**6**
8	**Wheelchair functionality**	**5**	**7**
8.a	Show diameter and highlight too narrow doors		5
8.b	Hide diameter and door highlight		2
9	**Create a dynamic list of elements**	**25**	**34**
9.a	Create class for elements list		3
9.b	Create parser for loading elements from the list		5
9.b.i	Function for loading predefined element		3
9.b.ii	Function for lading custom element		2
9.c	Create parser for saving default elements		1
9.d	Create parser for saving custom elements		6
9.d.i	Function for saving custom walls		4
9.d.ii	Function for saving custom elements		2
9.d.iii	Function for saving custom rooms		2
9.e	Dialog for saving custom elements		2
9.f	Dialog for insterting custom elements to the plan		2
9.g	Testing		15

Continued on the next page

ID	*Task*	*Estimated*	*Actual*
10	**Rotate one of two connected walls**	**15**	**17**
10.a	Function that allows for rotation		6
10.b	Create dialog for rotating wall		3
10.c	Testing		8
11	**User interface**	**20**	**22**
11.a	Create buttons on plan view		3
11.b	Create table with dynamic list of elements		6
11.c	Display plan information in the load plan screen		5
11.d	Write dimensions above the elements		5
11.e	Add zoom buttons		2
11.f	Remove decimal points from labels		1
12	**Write documentation**	**10**	**12**
Total hours		*110*	*135*

Table 9.1: Backlog for sprint 4

ID	*Task*	*Estimated*	*Actual*
1	Refactor code	5	5
2	Delete a plan from the device (and the list)	4	5
3	Export a plan to PDF, PNG or native plan format	8	11
4	Share plans over e-mail	10	9
5	Import plan for editing	4	4
6	Draw a room element with predefined dimensions	2	3
7	Lock the plan zoom and pan	2	6
8	Wheelchair functionality	5	7
9	Create a dynamic list of elements	25	34
10	Rotate one of two connected walls	15	17
11	User interface	20	22
12	Write documentation	10	12
13	Sprint planning	2	2
14	Sprint meetings	3	3
15	Customer meeting	6	6
16	Write report	79	57
Total hours		*200*	*203*

Table 9.2: Total work hours for sprint 4

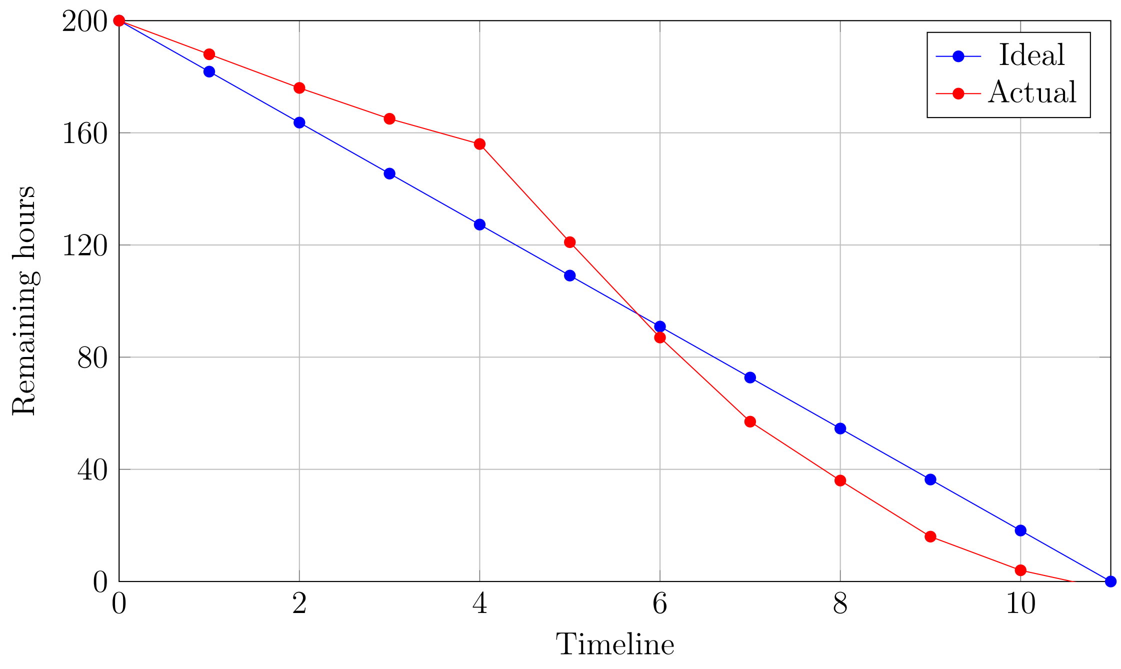

Figure 9.1: Burn down chart for sprint 4

9.5 Design and implementation

Deleting a plan is done after the delete button is clicked. The function takes the name of the plan, builds a file location from it, and the deletes the file. After that, the table of plans is refreshed.
Exporting a plan is done in the functions `-(void)btnExportPDFClicked :(id)sender` and `-(void)btnExportPNGClicked:(id)sender` of the `NVAPlanViewController`.
`NAVAppDelegate.m` takes care of importing a plan. A file is translated to a URL, and then using the storyboards, a new `ViewController` is created in order to display the plan.
A complete room can be drawn using the `touchesBegan` and `touchesEnded` methods. After the screen is touched, a dialog for entering dimensions appears, and after the user enters the data, the room will be drawn using four connected walls. This is handled by the selector `-(void)btnInsertRoomClicked:(id)sender`. Custom rooms can be saved for later use, and the dialog will not show when a saved room is placed on the plan.
Locking of the zoom level and panning is done when the respective buttons in the plan view are pressed. Their implementation is simple and does not require further explanation. The same applies to the buttons for zooming in and out as

they only change the `zoomScale` property of the `scrollView`.

Wheelchair functionality The `LongPressElement` gesture recognizer is expanded with code to recognize whether the element in question is a wheelchair. A long press on the wheelchair will go through the array of all doors in the plan and highlight all those whose width is narrower than the wheelchair's. In addition, a circle to represent the wheelchair rotation radius will be displayed. Another long press on the wheelchair will remove the circle and the highlight of the doors.
`-(NAVElementsList *)loadElementsList` loads elements from a list to the `TableView` that should hold them.
`-(void)createDefaultElements` is called from `NAVAppDelegate` and will create a list of default items if the application was started for the first time on the device.
`-(BOOL)SaveElementToListElementCategory:(NSString*)category attributes:(NSString*)attributes` saves custom elements to a list. The `(NSString*)category` can be one of three child categories: `"WallsChild"`, `"ElementsChild"` or `"RoomsChild"`. The `(NSString*)attributes` string holds the name of the element and its attributes (dimensions, diameter and/or label).
A dialog for rotating two connected walls will appear if a long press occurs on their mutual corner. Rotating of walls is done in the selector `btnChangeAngleClicked :(id)sender`, after showing another dialog for choosing the opposite angle or confirming the angle. The angle used for rotation is calculated, the end point of the wall to rotate is moved, and the wall layer is redrawn.

9.6 Testing

The main new functionality implemented in this sprint was adding custom elements, adding wheelchairs and checking for wheelchair accessability, deleting plans and the ability to export and share plans. In addition, steps to improve remedy the issues that were discovered with the usability of the application at the conclusion of the previous sprint were taken. We decided creating and using custom elements was functionality that was central to the usage of the application, as this would effectively replace importing items from the database, and thus gave these test cases a high priority. We also decided the same applied to exporting, sharing and deleting plans in addition to testing wheelchairs and wheelchair accessability. Creating rooms, horizontal/diagonal walls, copying elements, locking the plan-viewer and zooming using buttons were given a medium priority as they exist as a shortcut to improve the experience of the user.

9.6.1 Test cases

After the implementation phase of this sprint was completed, twelve tests were performed:

- Creating custom elements: *Test case INTG-41*
- Using saved custom elements: *Test case INTG-42*
- Copying elements: *Test case INTG-43*
- Creating horizontal/diagonal walls: *Test case INTG-44*
- Creating rooms: *Test case INTG-45*
- Exporting plans: *Test case INTG-46*
- Sharing plans: *Test case INTG-47*
- Deleting plans: *Test case INTG-48*
- Locking the plan zoom and pan: *Test case INTG-49*
- Zooming using buttons: *Test case INTG-410*
- Creating wheelchairs: *Test case INTG-411*
- Wheelchair accessability: *Test case INTG-412*

See appendix C.4 for a detailed summary of the test executions and results.

9.6.2 Test evaluation

Two of the integration tests in this sprint were unsuccessfull: deleting plans and sharing plans, while all of the other tests went without any problems.
When deleting plans, the plans were removed from the list of plans in the menu to open plans and seemingly deleted, but when exiting the menu and opening it again, the plan would reappear. We discovered that the xml plan file was not being deleted from the folder containing the plan files, and were able to fix the problem quite quickly.
When exporting plans, we were able to correctly send the plan file over e-mail and it was correctly received, but the recipient was not able to open the plan in their own *BoligApp* application. We suspect this is because the device did not recognize the plan file type, but were not able to remedy this problem before the end of the sprint.

9.7 Deliverables

At the end of this sprint, we were able to finish and present to the customer the following requirements:

- **FR5**: *Add custom elements*

9.8 Customer feedback

As this was the last sprint the application demonstrated at the meeting held with the customer was one of the last versions of the application that we would be able to show before delivering the project. This was understood by the customer, and the product delivered at the conclusion of this sprint therefore represented a final draft of the application, with the possibility of making some minor adjustments to the program in the closing stages of the project.
Like in the previous sprint the customer tested the application and performed some typical usage scenarioes. Overall the user experience of the customer was greatly improved as we had addressed all of the issues detected during the previous trial, and added shortcuts for creating square rooms in addition to taking measures to avoid make it as easy as possible to input elements.
A few issues the customer pointed out were that the created custom items were displayed in the menu to add items as "custom element" suffixed with their dimensions. The customer would have preferred the item to be displayed as their given labels. We had also not had time to translate the application to Norwegian, as half of the development team does not speak Norwegian.
Overall the customer seemed very pleased with the product we had produced, given the limitations of this project (the most prominent of these being the time available), and felt we had produced something that could be useful as a tool for their organization.

9.9 Sprint evaluation

This section will take a look at things we have accomplished and include a quick summary of what went well and what didn't go so well during this sprint.

9.9.1 Positives

The presentation and user testing of the application to the customer went without any problems this time. We had a very large backlog for this sprint, especially in terms of items to implement in the application, and are very pleased that we managed to implement all of the items from the backlog.

9.9.2 Negatives

Because of the large amount of backlog items in this sprint related to implementation, we did not do enough work on documentation and writing the project report as we should have, and were at the conclusion of this sprint quite far behind schedule in these two areas.
Although the presentation of the application went well, we would have liked to be able to show the customer an application that was fully translated, had label names for custom elements displayed in the menu to create new custom elements, and in general was a bit more polished.

Part III

Conclusion and evaluation

Chapter 10

System architecture

This chapter will describe the system architecture. It will explain the main system components and their mutual interaction. The view model that we will use is Philippe Kruchten's 4+1 view model [6]. The 4+1 view architecture is represented by logical, process, implementation and deployment views. Each of these consists of different types of diagrams which describes the system from the perspective of a particular stakeholder (end user, developers, etc. In general, a stakeholder is a person who has influence on the system)

10.1 Logical view

In this view, the application's functionality in terms of structural elements, key abstractions and mechanisms will be shown. Based on this view, a functional analysis is made.
For documenting the logical view we have used class and package diagrams.

10.1.1 Class diagram

We will represent both the back end and the front end of our application using a class diagram, where only important classes and methods are included for readability. See figure 10.1 for the class diagram. The data for our application is represented by `NAVElement`, `NAVPlan`, `NAVWall` and `NAVRectangularElement`. The class `NAVParser` is also a part of the data management, and it implements parsers for saving the list of plans and elements to an XML file. Using the `TBXL` class, `NAVParser` is able to read from an XML file. `NAVPlan` is our main data class. Its instance contains objects of the type of its child classes (`NAVWall` and `NAVRectangularElement`), and represents the elements in the plan.
`NAVElementsList` will hold a list of all elements that can be added to the plan,

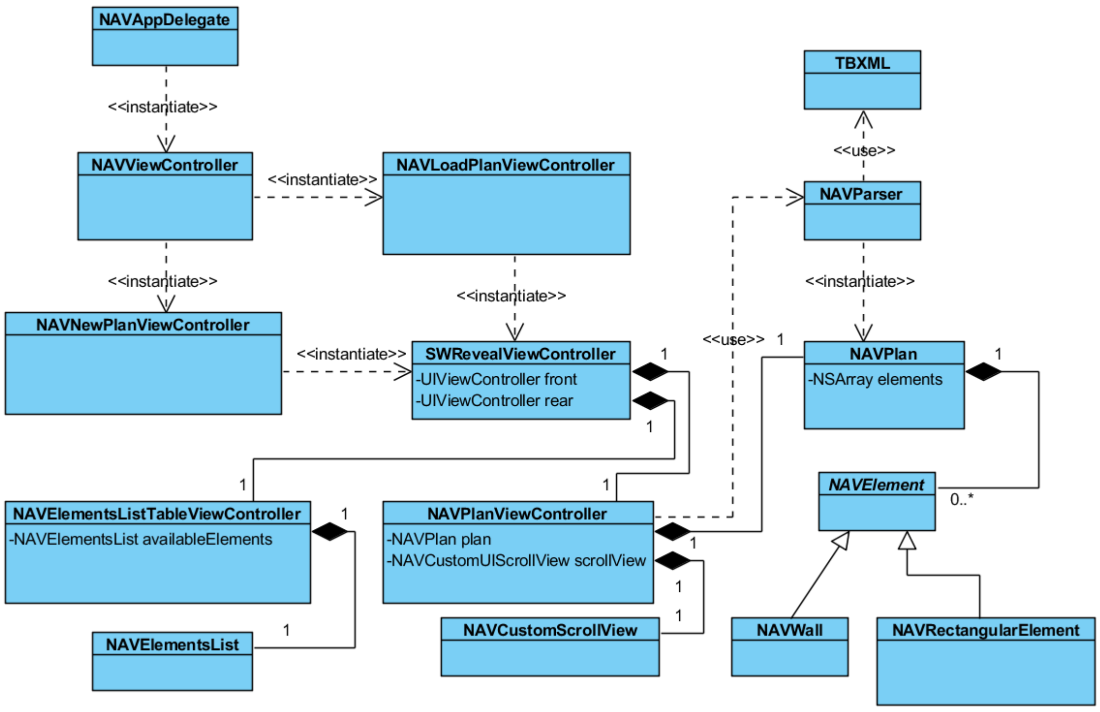

Figure 10.1: Class diagram

the default one and the one that was added by the end user.
`NAVCustomScrollView` is our main view for displaying a plan. We have extended `UIScrollView` with methods and properties to suit our needs. It will hold all objects which represents elements in a plan, and methods for manipulating those elements.
`NAVViewController` is the start view of our application, and it works like a main menu of the application. From it, the end user can switch to `NAVLoadPLanViewController` which has list of all possible plans, or the user can navigate to `NAVNewPlanViewController` on which data for a plan can be entered and a new plan can be created from it.
`NAVPlanViewController` is used to display plans. It contains an instance of `NAVCustomScrollView`, and is the parent view controller of `NAVElementsListTableViewControler`.

10.1.2 Package diagram

The organisation of packages is reflected in figure 10.2. A package diagram is often use to organise use case and class diagrams.

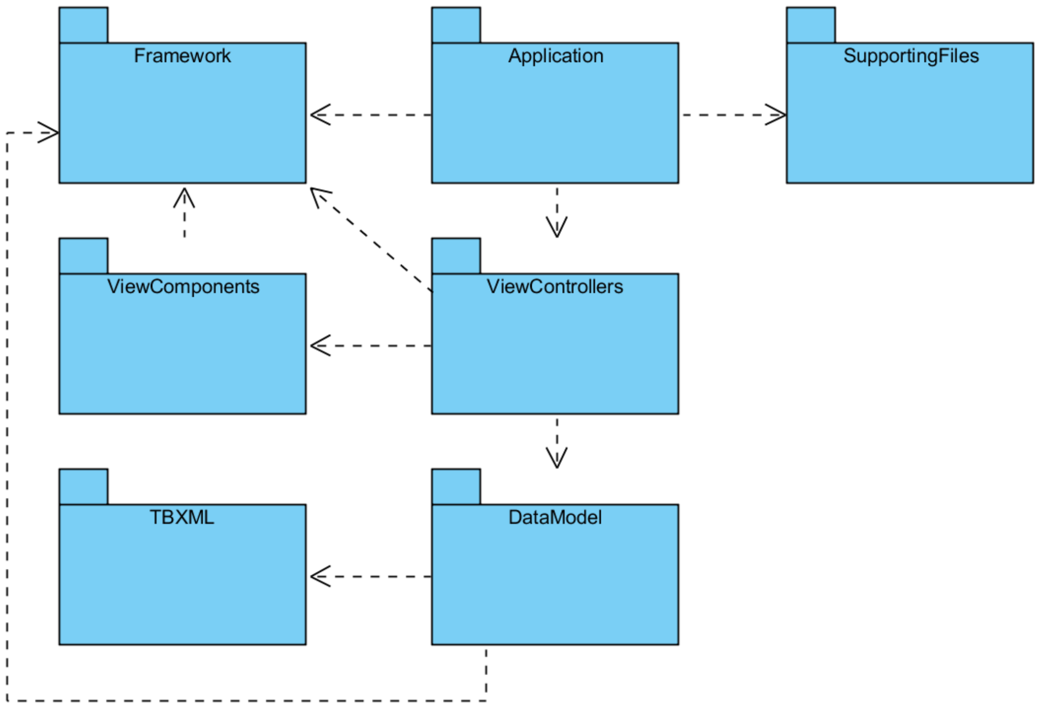

Figure 10.2: Package diagram

10.2 Process view

Non-functional aspects like performance and scalability are considered in the process view. The focus in the process view is how the system is acting at runtime. We used sequence and communication diagrams.
Sequence diagrams shows the sequence of messages passed between the objects on a vertical timeline. Communication diagrams shows communications between objects at runtime during a collaboration instance. Sequence diagrams are focused on the flow of messages throughout an interaction over timeline, communication diagrams are focused on relations between participants.

The diagrams 10.3 and 10.4 show the process of creating a new plan. When the user taps on the button, an event for tapping the button is registered. The main menu view, which represents the user interface, will register the action and send a message to its view controller. The main menu view controller will instantiate a new Plan view controller with its own view, as a user interface representation. Then the user can enter the required data and tap the button for creating a new plan. In response to this action, the system will instantiate a Plan view controller. The Plan controller will create a Plan model, and afterwards it will open it in the

Plan view and show the data to the user.

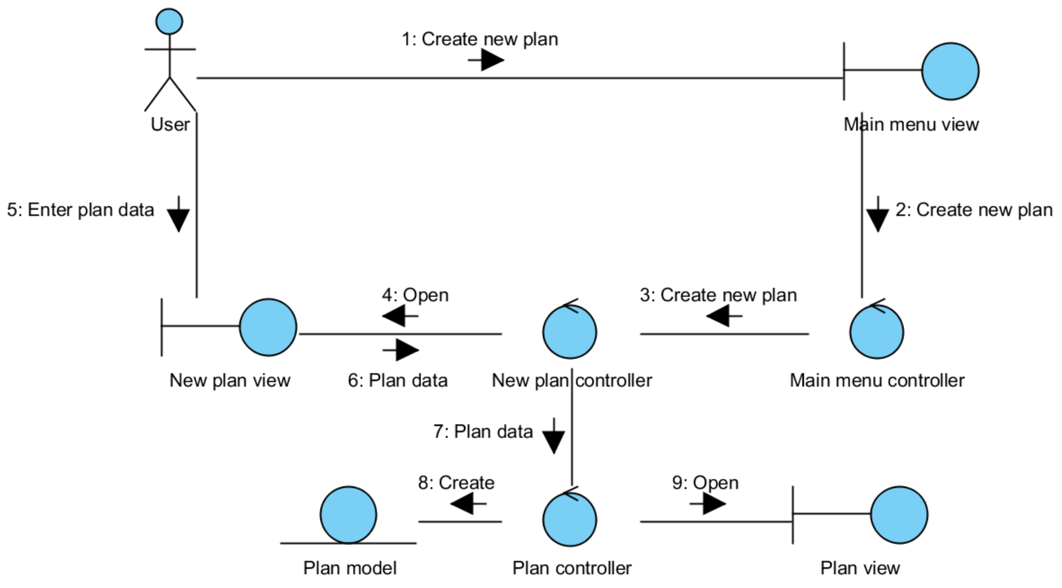

Figure 10.3: Communication diagram for creating a new plan

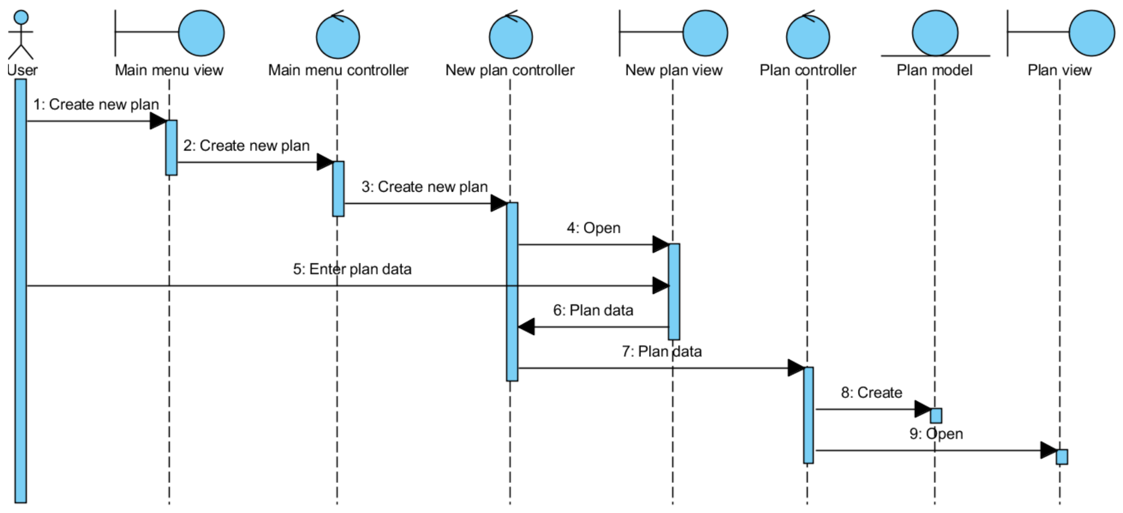

Figure 10.4: Sequence diagram for creating a new plan

The diagrams 10.5 and 10.6 show the process of loading a plan. When the user taps on the button, he is brought to the load plan view. Then user can select which plan to load and tap the button for opening the plan. In response to this action, the system will instantiate a Plan view controller. The Plan controller will create a Plan model, and afterwards it will open it in the Plan view and show the data to the user.

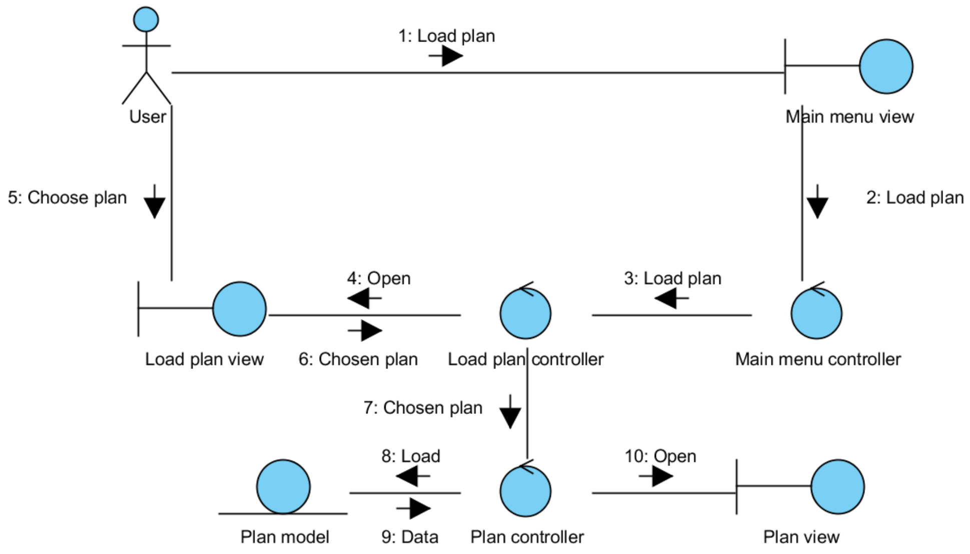

Figure 10.5: Communication diagram for loading a plan

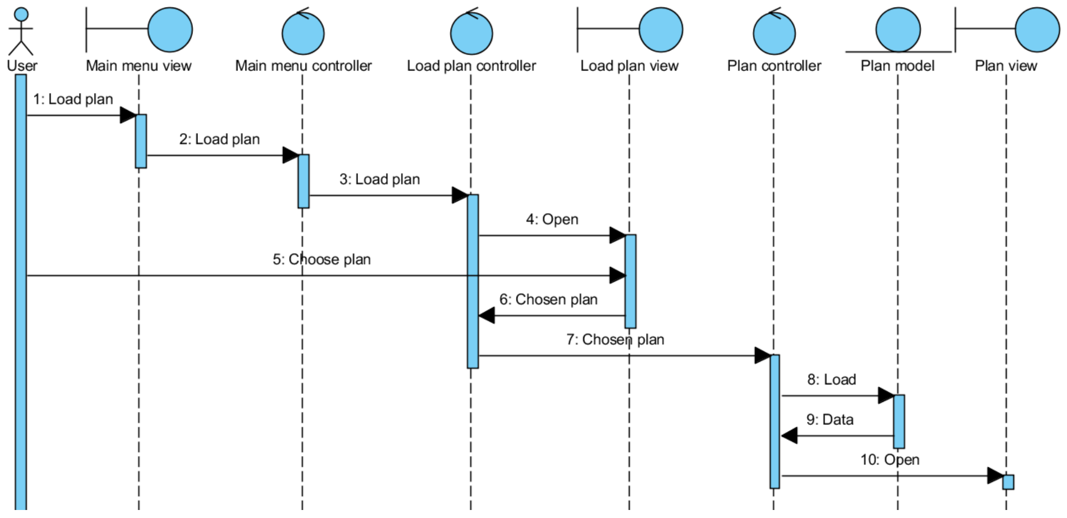

Figure 10.6: Sequence diagram for loading a plan

If the user wants to add an element, he interacts with the Plan view which represents the user interface. User actions are forwarded to the Plan controller, which checks for constraints and adds the element to the plan model. If the model data is changed, the controller will update the Plan view to show the changes. See diagrams 10.7 and 10.8.

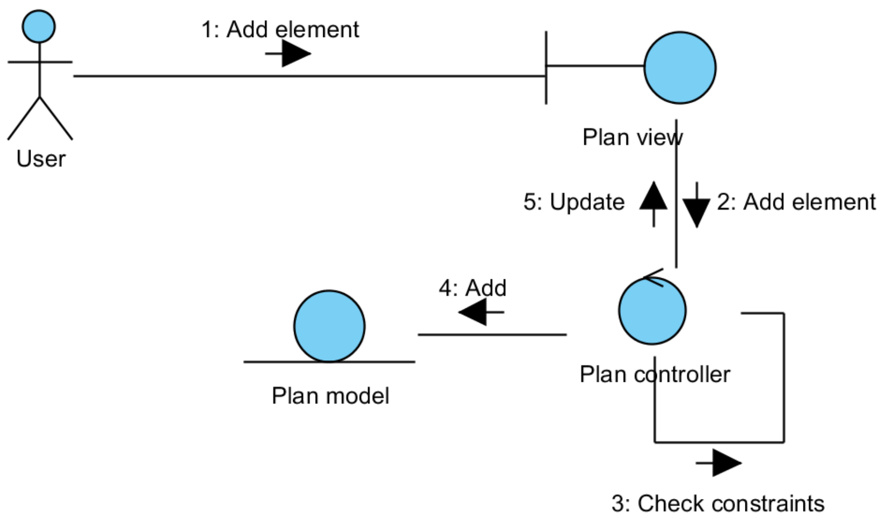

Figure 10.7: Communication diagram for adding an element

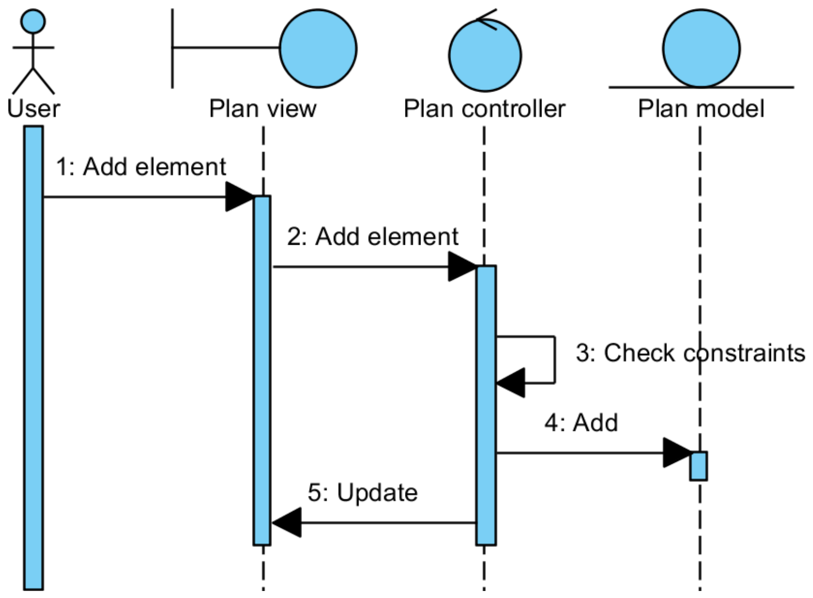

Figure 10.8: Sequence diagram for adding an element

If the user wants to edit a plan, move elements or rotate and resize them, he interacts with the Plan view. User actions are forwarded to Plan controller, which checks for constraints and edits data in the Plan model. If the model data is changed, the controller will update the Plan view to show the changes. See diagrams 10.9 and 10.10.

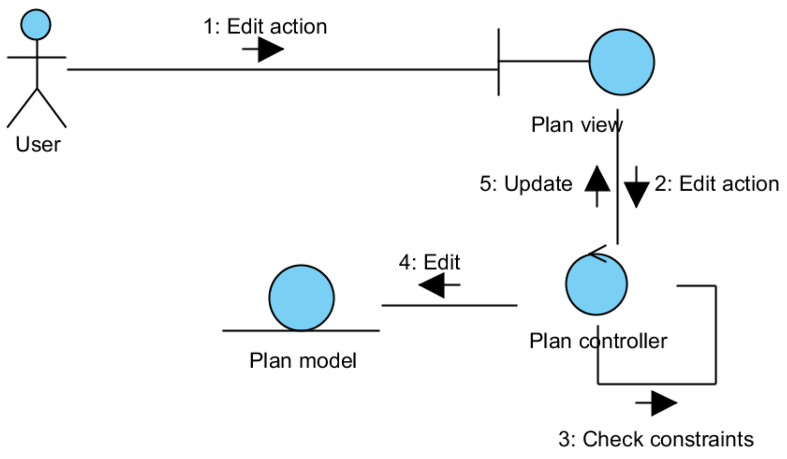

Figure 10.9: Communication diagram for editing a plan

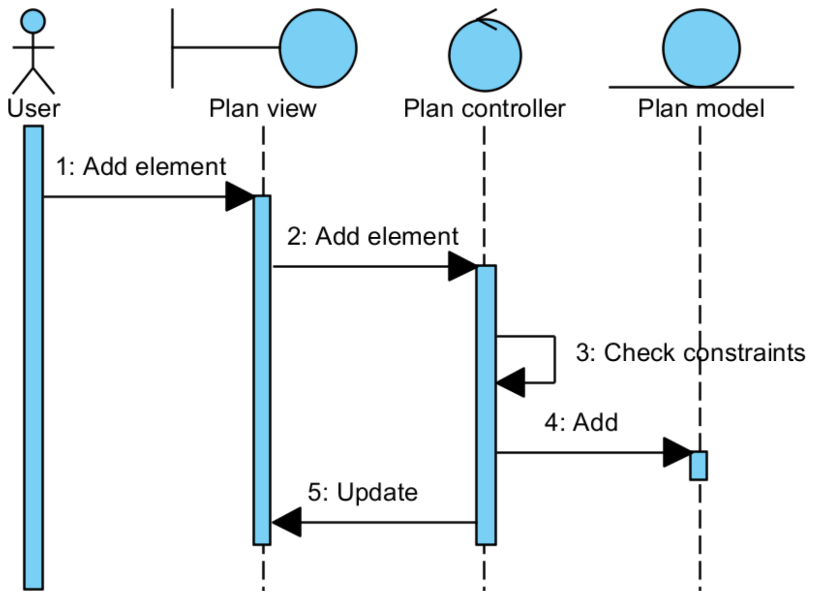

Figure 10.10: Sequence diagram for editing a plan

If the user wants to delete an element, he interacts with Plan view. User actions are forwarded to the Plan controller, which removes the element from the Plan model. After changes has occurred, the controller will update the Plan view to show the new state. Se diagrams 10.11 and 10.12.

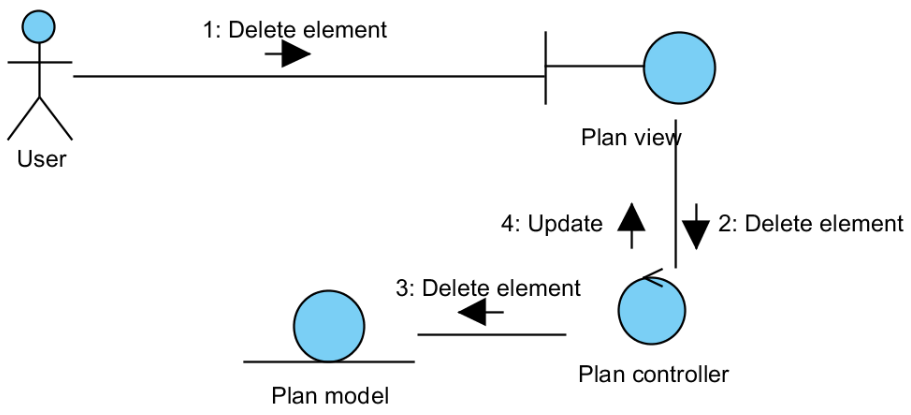

Figure 10.11: Communication diagram for deleting an element

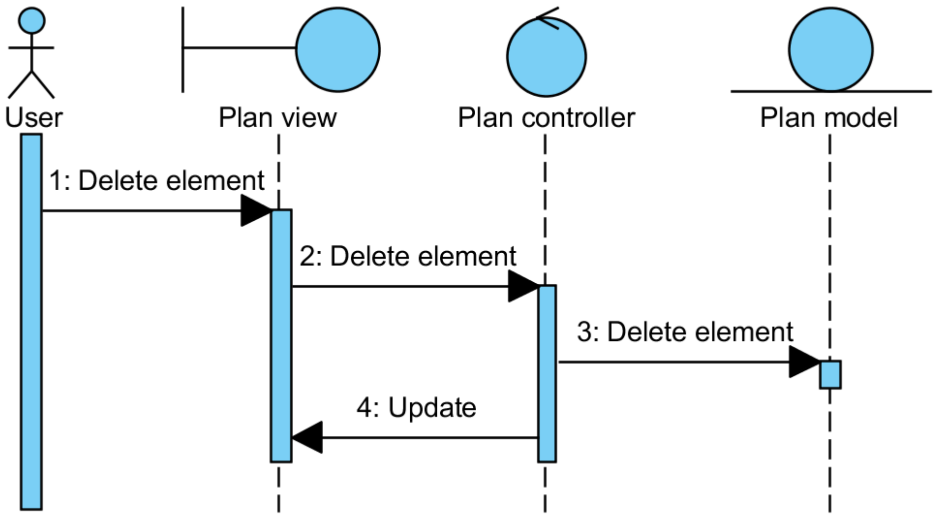

Figure 10.12: Sequence diagram for deleting an element

If the user wants to save a plan, he interacts with the Plan view. User save actions are forwarded to the Plan controller, which will save the plan model. The Plan controller will notify the user that the plan is saved. See diagrams 10.13 and 10.14.

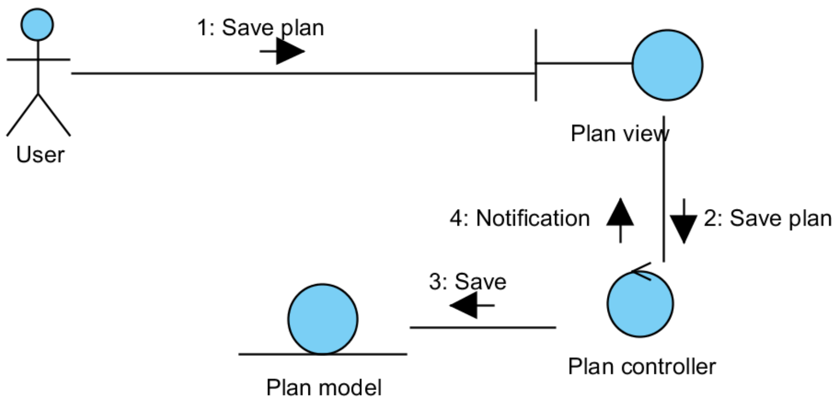

Figure 10.13: Communication diagram for saving a plan

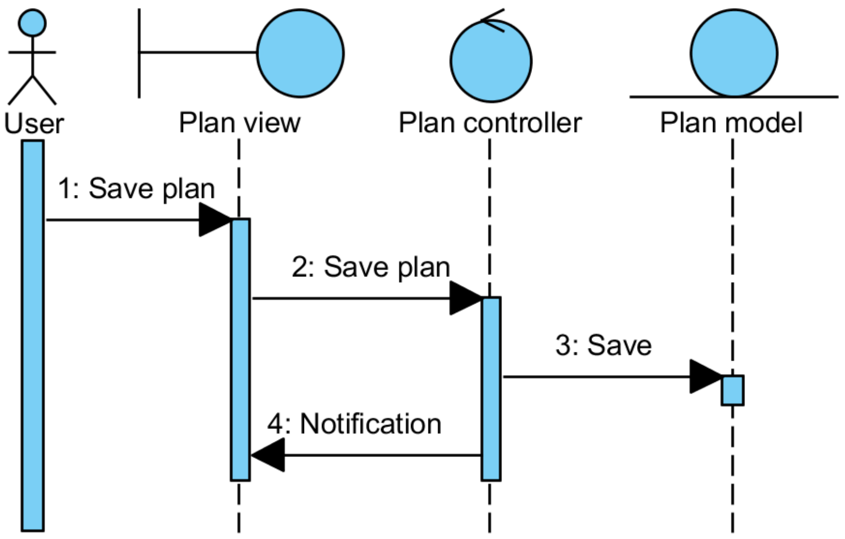

Figure 10.14: Sequence diagram for saving a plan

After an export action is initiated, an action is sent to the Plan view. It is forwarded to the Plan controller, which will send an action to the Schematic plan to create a plan. The Plan controller will notify the user when the export is finished. See diagrams 10.15 and 10.16.

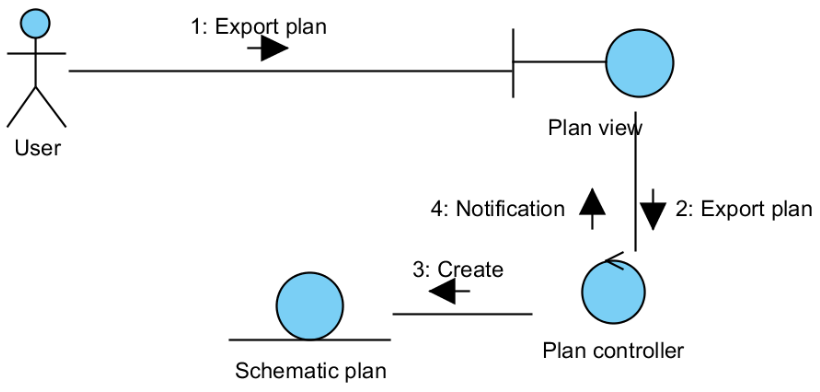

Figure 10.15: Communication diagram for exporting a plan

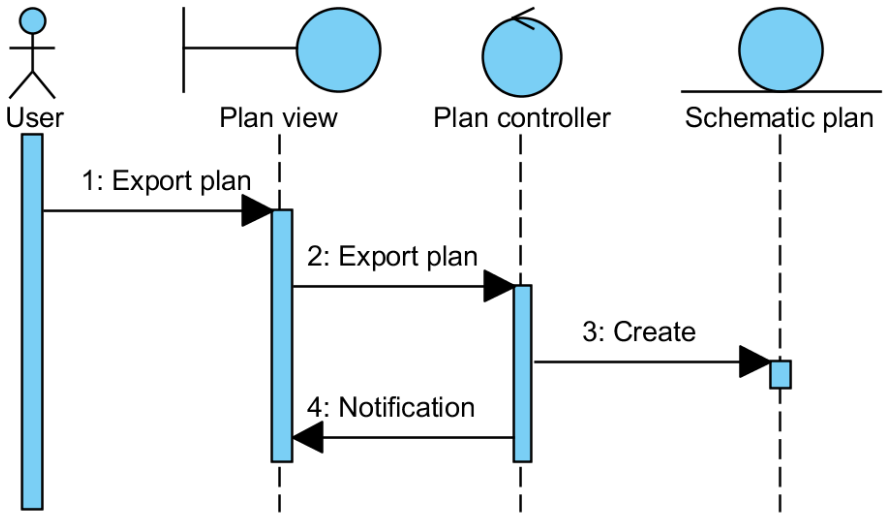

Figure 10.16: Sequence diagram for exporting a plan

After a share action is initiated, an action is sent to the Plan view. It is forwarded to the Plan controller, which will send an action to the Schematic plan to create a plan. The created plan is sent back to the Plan controller. A message is sent to the user that a plan can be shared. The Plan controller will notify the user when export is finished. See diagrams 10.17 and 10.18.

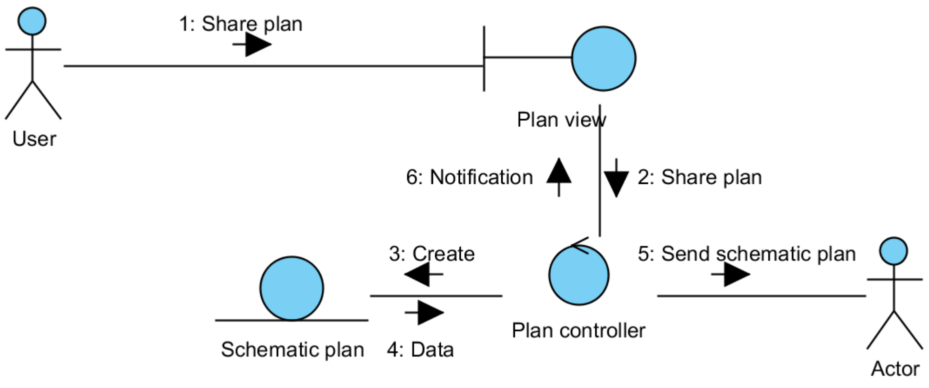

Figure 10.17: Communication diagram for sharing a plan

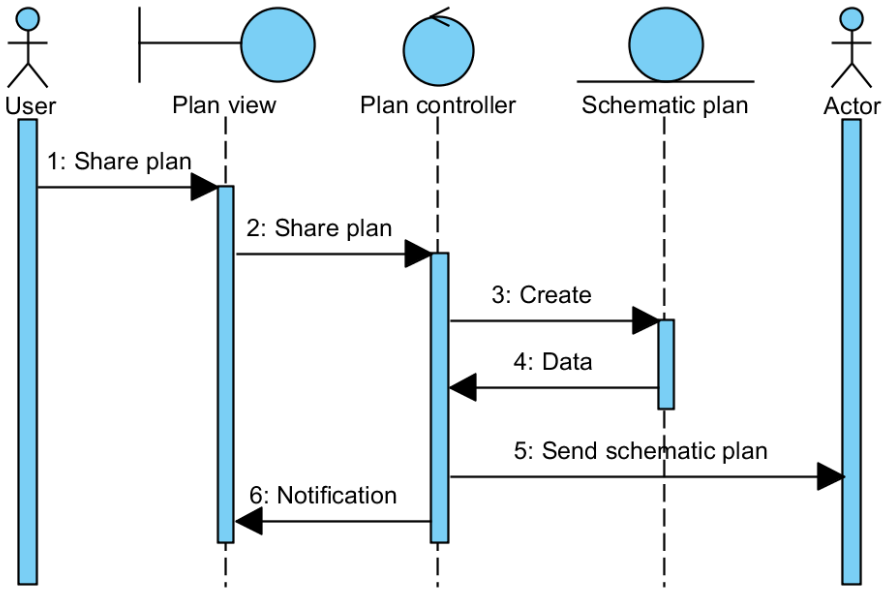

Figure 10.18: Sequence diagram for sharing a plan

10.3 Implementation view

This view focuses on system architecture and components that are used for assembling and releasing the system. It will focus on actual software module organization in the development environment. The diagrams in this view represent physical level artifacts that are built by the team.
For that representation we have used a components diagram.
Interface represents the user interface components. Plan controller represents the part of the application that is responsible for interaction with the plan. Plan model is connected to Plan controller as it represents the plan data. Our application is based on the UIKit and CoreData frameworks which are core components of the iOS SDK. The TBXML component is responsible for parsing data from and to XML files. See the component diagram (figure 10.19).

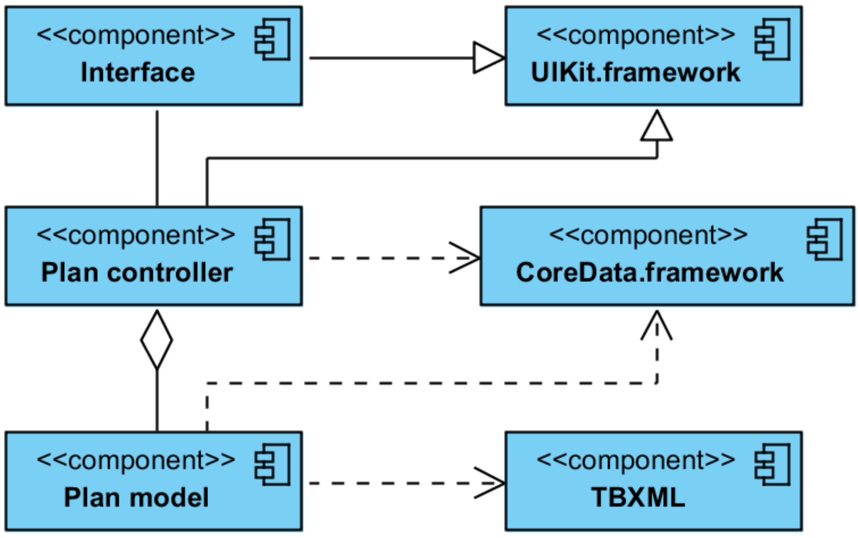

Figure 10.19: Component diagram

10.4 Deployment or physical view

The deployment view represents the hardware on which the system is executed. It will show physical limitations for distributing the system. It provides hardware configurations and maps the components from the Implementation view to these configurations. In our case, our application is made for a specific device, the iPad, which was specified by the customer. See the deployment diagram (figure 10.20).

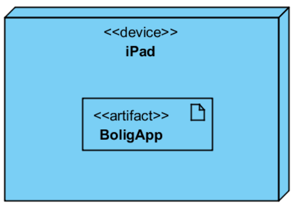

Figure 10.20: Deployment diagram

10.5 Use case view

This view will describe use cases that explains the system from the perspective of the end users and other stakeholders. The use case view should be the first view that is created in the system development lifecycle. The use case view forms the reason why all other views exist. It is represented by Use Case diagrams, which consists of actors and actions. See the Use Case diagram (figure 10.21).

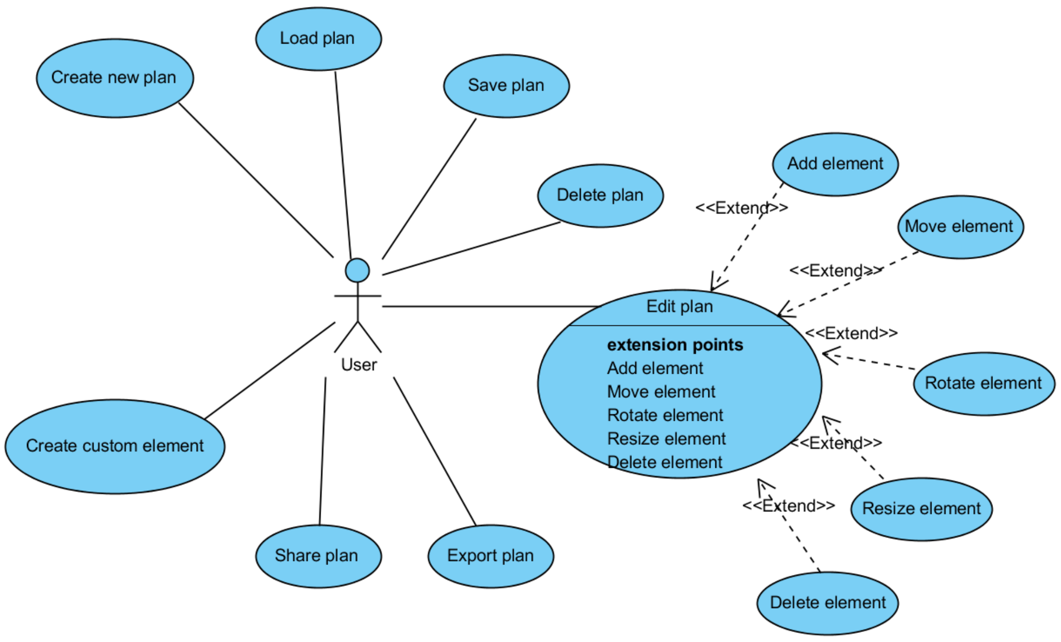

Figure 10.21: Use case diagram

Chapter 11

Acceptance testing

In this chapter we will perform an evaluation of the functionality of the program to determine if the requirements specified in section 5 are adequately fulfilled. This will be done by performing system tests that cover all of the implemented functional requirements. Requirements that we did not attempt to implement will not be tested.

11.1 Functional requirement coverage

Table 11.1 gives an overview of which functional requirements were covered in the implementation, and which were not. This section will highlight which functional requirements were not covered during development, and why.

Adding database elements

As described in section 9.3, there were some internal communication problems within NAV that lead to us neither being able to get the necessary information about NAV's assistive technology database nor the permission to read data from the database within a time frame that allowed us a realistic chance of implementing the functionality.

Camera measurement

The ability to draw a schematic drawing of a room using the panoramic photo option of a mobile device was a requirement that we spent a lot of effort on researching in the initial stages of this project. Already in the first meeting with the customer, we made it clear that we feared implementing such a functionality was an unrealistic requirement for a project of this magnitude. We searched online, and tried to find help at NTNU's department of computer science. When this lead

Requirements covered	Sprint implemented
FR1 *Create new plan*	3
FR2 *Open plan*	1
FR3 *Save plan*	3
FR4 *Add walls and other elements*	3
FR5 *Add custom elements*	4
FR7 *Move elements*	2
FR8 *Resize elements*	2
FR9 *Rotate elements*	2
FR10 *Remove elements*	2
FR11 *Display dimensions*	3
FR12 *Detect collision between elements*	2
FR14 *Zoom*	3
FR15 *Pan*	3
FR16 *Export*	3
Requirements not covered	
FR6 *Add database elements*	
FR13 *Camera measurement*	

Table 11.1: Functional requirements implemented

nowhere, we decided to abandon implementing this requirement out of fear of completely bottlenecking the whole project. This was made clear to the customer prior to development began.

11.2 Requirement testing

We will test that the requirements are fulfilled by using system testing. As specified in the test plan (see section 2.5), the scope of each system test will be a set of requirements. The individual test cases will be formulated as relatively complex and high-level tasks to perform in the application. Success criterias will be relatively vague, because we will continuously be monitoring program behaviour throughout the tests and the amount of specific things required for a test to succeed will be too much to write in a test case cell.

11.3 Test cases

We have formulated four test cases to test all of the requirements that have been implemented:

- Create plan: *Test case SYST-1*
- Populate plan: *Test case SYST-2*
- Edit plan : *Test case SYST-3*
- Export and delete plan: *Test case SYST-4*

See appendix C.5 for a detailed summary of the test executions and results.

11.4 Evaluation

All of our system tests were successfull. Based on this, we determine that our application adequately covers the requirements and is of acceptable quality.

Chapter 12

System evaluation

In this chapter we will perform an evaluation of the final system. We will focus on the following subset of the characteristics of software quality defined by the ISO 9126 standard [11] for software quality:

- Functionality
- Efficiency
- Usability

Efficiency and usability are also qualities that we have listed as non-functional requirements for our application. This will be done by asking subjects to perform a typical usage scenario of the application that we will define. The subjects will be given the user manual to the application(see appendix B), but will not receive any assistance while performing the usage scenarioes. We will in other words be asking these subject users to perform scenario tests for us, but this process will not be documented as normal tests, as the focus area is not whether the system is technically functioning properly, but how a user experiences the usage of the system.
After we have gathered the data from the user tests, we will use the data gathered to perform an evaluation of the system.

12.1 Usage scenario

Here we will define the usage scenario that we will ask the users to perform. This scenario will be conveyed to the user as a sequence of instructions. These instructions are specified as high-level tasks and are formulated without containing specific buttons to press or any other application-specific instructions to avoid giving the subjects too much information, because an important quality we are

Task ID	*Task description*
T1	Open the application
T2	Create a new plan
T3	Draw a plan of your own bedroom including all furniture as accurately as possible
T4	Save the plan
T5	Exit to the main menu
T6	Open the plan you created
T7	Send a picture of the plan you created as an e-mail to yourself

Table 12.1: Task items given to subjects for usage testing

testing is the intuitiveness and ease-of-use of the application. Table 12.1 contains this set of instructions that is given to the subjects.

12.2 User questionnaire

We decided to create a questionnaire using Google Forms [1]. The first page of our questionnaire gathered a little bit of personal information about each subject and some information about their familarity with using iPad applications. We then gathered information about the subject's experience of using the application by listing a series of statements, and having the subject enter on a scale of 1 to 5 how much they agreed with the different statements. We had one page for each screen of the application, with the page listing statements about the plan-viewer being the most extensive.
Table 12.2 shows all the questions given to each user in the online questionnaire. The Horizontal lines represented the different pages of the questionnaire. Each page of the questionnaire covered a different part of the application, and included a description to the user to clarify what the questions referred to. For example the screen with questions about the window for drawing a plan, included a description that clarified that the window for drawing a plan referred to the part of the plan viewer where the actual and elements were shown, but did not refer to the sidebar for adding elements to the plan.
The last page of the questionnaire asked the subjects if they had any comments about how they experienced the usage of the application.

[1]`https://support.google.com/drive/answer/87809?hl=no`

Question ID	*Question*
Q1	What is your age?
Q2	What is your gender?
Q3	Do you own and frequently use an iPad or iPhone?
Q4	On a scale of 1-5 how proficient do you consider yourself with using iOS applications?
Statement ID	*Statement*
S1	I thought the main menu was intuitive and easy to understand
S2	I thought the main menu looked good
S3	I thought the main menu included all the functionality I required
S4	I thought the screen for creating a new plan was intuitive and easy to understand
S5	I thought the screen for creating a new plan looked good
S6	I thought the screen for creating a new plan included all the functionality I required
S7	I thought the window for drawing a plan was intuitive and easy to understand
S8	I thought the window for drawing a plan looked good
S9	I thought the window for drawing a plan included all the functionality I required
S10	I thought drawing a room was easy and satisfying
S11	I thought populating a room with items was easy and satisfying
S12	I thought it was easy to understand how to export the plan
S13	I thought the ability to lock panning and zooming in the plan viewer was helpful
S14	I thought the buttons to zoom in and out in the plan viewer were helpful
S15	I thought the sidebar for adding elements in the window for drawing a plan was intuitive and easy to understand
S16	I thought the sidebar for adding elements in the window for drawing a plan looked good
S17	I thought the sidebar for adding elements in the window for drawing a plan included all the functionality I required
S18	I thought the screen for loading plans was intuitive and easy to understand
S19	I thought the screen for loadnig plans looked good
S20	I thought the scren for loading plans included all the functionality I required

Table 12.2: Questions and statements in questionnaire given to each test subject

12.3 Questionnaire results

We conducted usage testing with 10 test subjects. Table 12.3 shows the results from this questionnaire.
Table 12.4 contains the comments that the subjects had about the usage of the application.

12.4 Questionnaire data analysis

12.4.1 Question grouping

By quality

To evaluate the program by the criteria specified in the beginning of the chapter, we have grouped the questions by which quality they describe as shown in table 12.5.

By program part

To evaluate the different parts of the program individually, we have grouped the questions by which part of the program the describe the quality of, as shown in table 12.6.

12.5 Conclusion

As shown in table 12.7, users thought our application had a functionality of 3.9, a usability of 3.8 and an efficiency of 4.0. These are scores that we are pleased with.
As shown in table 12.8, users gave our main menu a score of 4.0, our new plan menu a score of 3.8, our load plan menu a score of 4.5, our plan drawing window a score of 3.5 and our sidebar for adding elements a score of 3.6. We would have liked to have received a better score the parts of our application that provide the functionality for drawing plans and pupulating plans with elements, while we are pleased with the scores our menus received.

User ID	*U1*	*U2*	*U3*	*U4*	*U5*	*U6*	*U7*	*U8*	*U9*	*U10*	
Query	*Results*										*Average*
Q1	21	23	20	22	19	20	56	49	22	24	27.6
Q2	M	M	M	M	M	F	M	F	M	F	N/A
Q3	Y	Y	N	Y	N	N	Y	N	N	Y	N/A
Q4	4	5	3	4	2	4	3	2	3	4	3.4
S1	5	3	5	5	4	3	5	3	5	5	4.2
S2	4	3	5	4	4	3	4	5	5	5	4.0
S3	3	3	4	4	4	3	4	4	5	4	3.7
S4	4	2	5	4	4	4	4	2	5	4	3.8
S5	4	3	4	4	3	3	3	3	4	4	3.5
S6	5	4	4	5	5	3	5	4	5	4	4.4
S7	3	3	2	4	4	3	4	3	3	3	3.2
S8	3	3	4	4	3	3	4	3	4	4	3.5
S9	5	3	4	4	3	3	3	4	4	4	3.7
S10	4	2	5	4	4	4	4	2	5	4	3.8
S11	3	3	2	3	2	3	4	3	3	3	2.9
S12	5	4	3	4	4	5	4	4	5	4	4.2
S13	5	5	5	4	4	5	4	4	5	3	4.4
S14	5	5	5	4	4	5	4	4	5	4	4.5
S15	4	3	5	4	4	5	4	4	4	4	4.1
S16	4	3	3	3	4	3	4	3	5	4	3.5
S17	4	3	3	3	4	3	4	1	3	4	3.2
S18	5	5	5	3	5	5	5	5	5	4	4.7
S19	5	4	5	3	5	5	4	5	4	4	4.4
S20	5	5	5	3	5	5	4	5	4	4	4.5

Table 12.3: Results from user questionnaire after performing usage testing

User ID	*Comment*
U1	Everything went better than expected
U2	Nice app!
U3	Strange that the settings button in the main menu didn't work.
U4	None
U5	I thought drawing a plan was kind of annoying as performing actions on elements didn't behave as I expected.
U6	What's the difference between export and share?
U7	None
U8	None
U9	None
U10	None

Table 12.4: Comments given by the different test subjects after performing usage testing

Functionality	*Usability*	*Efficiency*
S3	S1	S3
S6	S2	S6
S9	S4	S9
S17	S5	S13
S20	S7	S14
	S8	S17
	S10	S20
	S11	
	S12	
	S13	
	S14	
	S15	
	S16	
	S18	

Table 12.5: Grouping of questions by program quality they describe

Main menu	*New plan*	*Load plan*	*Draw plan*	*Add element*
S1	S4	S18	S7	S11
S2	S5	S19	S8	S15
S3	S6	S20	S9	S16
			10	S17
			S11	

Table 12.6: Grouping of questions by program part they describe

Software quality	*Average score*
Functionality	3.9
Usability	3.8
Efficiency	4.0

Table 12.7: Scores of software qualities given by users

Software quality	*Average score*
Main menu	4.0
New plan	3.8
Load plan	4.5
Draw plan	3.5
Add elements	3.6

Table 12.8: Scores of program parts given by users

Chapter 13

Project evaluation

In this chapter we will discuss our experience with this project. We will present our thoughts about using scrum, and how the group worked together. We will also discuss the risks described in section 2.3, explain their impact on the project and how the preventive measures worked. The management of time is an important aspect of this course, as the group has to constantly balance how much they work on the product and the report. Lastly, we will give our evaluation of the course.

13.1 Using scrum

The essence of the scrum methodology is to re-evaluate the project at fixed intervals (sprints). This makes it an ideal work flow if the project is not very accurately defined, and if some tasks are not sure to be finished as planned. In our project, both of these are the case. The customer had some ideas on what functionality they wanted in the product, but most of the design decisions were up to us. Some of the functionality that we had implemented in one sprint were, after feedback from the customer, improved and expanded in later sprints. For example, the functionality for adding elements, that was implemented in sprint 1, was expanded throughout the sprints until, in sprint 4, it included adding custom elements.
For the last sprint, we had initially planned to connect the application to NAV's database of assistive technologies. Because of some communication problems, however, as explained in section 9.3, we obviously had to re-structure our plan for sprint 4. Using the sprint planning meeting, it was easy to adapt to the problem, and plan new tasks to work on.
As a team, we have worked very closely, so that we have usually known what the other team members have been working on. This made the scrum stand-up meetings seem quite rigid and unnecessary, although we carried them through as a formality.

13.2 Risk evaluation

This section discusses the items we described as possible risks in table 2.1. We will look at the preventive measures described, if we properly executed these preventive measures, and if they were effective. If a risk occurred, we will discuss the problems it caused and the consequences these entailed.

13.2.1 Unfamiliar with development platform

This case was not much of a risk as it was a fact we knew would be problematic, as only one of our group members had any previous experience with using the development platform we would need to use.
The preventive measures we had described were to read a lot of documentation, try to find people with knowledge that could assist us at the university, and work through tutorials. These preventive measures were effective. We allocated enough time at the beginning of our project for learning about the development platform we would use, and this risk did not become a significant problem.

13.2.2 Poor time planning

We made good plans for the progress of the implementation early on in the project. This plan was followed quite closely up until the 4. sprint when we encountered the problem with the database communication (see section 9.3), and we had to reschedule our plan. This was all according to the preventive actions defined.
The report writing, on the other hand, was not as closely planned, and we ended up doing a lot of work in the last week.

13.2.3 Customer changes their mind

This risk did not occur, as the customer was clear and specific throughout the course of this project about what they wanted us to develop. We executed our preventive measure throughout the project of maintaining frequent contact with the customer, but did not see if this measure was effective as the problem did not occur.

13.2.4 Customer does not have technical expertise

Like the group being unfamiliar with the development platform, the customer not having technical expertise was a problem that we knew we would need to handle. The preventive measures we had described were to have early contact with the customer. We had extensive early contact with the customer, to make sure that we

had a clear set of requirements for the application that was realistic to implement in the given time period.
Despite us executing the preventive measure we had described, this risk became a problem at a later point during the development process when we needed access to a database to connect it to our application. The problem was that the database is managed by an external party, and communication between us and this party took place through the customer. This resulted in the wrong questions being asked to the about the database system to the database responsible, and contributed to the problems we had with connecting to the database system.

13.2.5 Missing equipment

This risk occurred early on in the project, as we realized during planning that we needed OSX machines to develop applications for iOS. We executed our preventive measure of quickly talking to the customer about the necessary equipment, and the customer quickly acquired the necessary equipment for us. Therefore, this risk occured but did not present a significant problem.

13.2.6 The scope of the project is too big

This risk was linked to the risk specifying that the customer did not have technical expertise, because both of them could result in the project having requirements that were unrealistic to implement. We executed the preventive measure of planning properly early on in the project and communicating clearly to the customer what we felt was realstic to achieve. This measure was effective and the risk did not become a problem.

13.2.7 Unexpected team member absence

This problem occured in sprint 2 (see section 7.9.2), when one of the team members was ill for almost a week. The person who was ill was kept up to speed on the progress of the team, but missed two scrum meetings. This problem resulted in the team being slightly delayed, but the severity of the problem was not that high and we quickly got back up to speed when the person returned.

13.2.8 Loss of data

We encountered a few cases throughout the course of this project where we lost data due, both in the code of the application and in the report. The use of a software repository handled these problems well, as we could look at the commit history and recover the data, and the only consequence of the problem was wasting time.

13.2.9 Difficulty with implementation

We did a thorough job in the planning and learning phase, as specified in the preventive measure for this risk, and did not encounter any major difficulties during the development of our application.

13.2.10 Technical problems with tools and software

As described in subsection 13.2.8, we had a problem where lost data a couple of times. This was due to a text editor automatically overwriting changes that had been pulled from the software repository. This problem could have been avoided by more carefully evaluating which software to use, as we stated as a preventive measure for this risk. The consequence of this problem was that we lost a couple of hours, until we found the source of the problem.

13.2.11 Internal conflict

We did not experience any significant internal conflicts throughout the course of this project. The preventive measures listed here executed, and were effective for keeping the group harmonious and morale good.

13.3 Time management

According to the course compendium, the total amount of work hours for our group should be $25 * 14 * 4 = 1400$. 25 person-hours per week seemed too much at first, but wee soon realized that this project would demand a lot of effort.
The period before the first sprint was used for preliminary study and the initial project planning. We wanted to be well prepared for the project, and put a lot of effort into this phase. During the sprints we tried to use equal amounts of time on the implementation and the report. The last period after the last sprint was focused mainly on writing the report, but a little implementation was done there as well. This was unquestionably the period with the most work hours, as the project was approaching the end. See figure 13.1 for an illustration of the distribution of work hours per task. We ended up working for a total of 1249 hours on this project, which is about 22.3 hours per person per week.

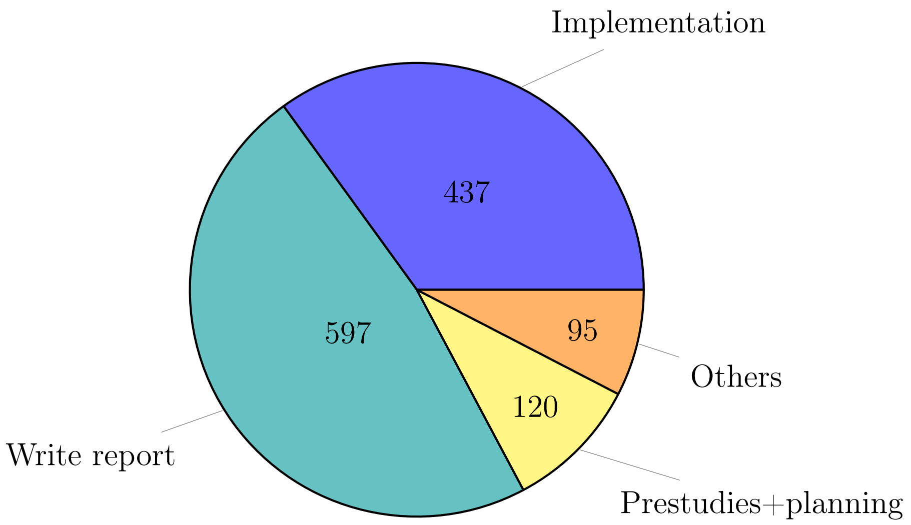

Figure 13.1: Pie chart illustrating the distribution of work hours per task

13.4 Group dynamics

Here we will evaulate how the group functioned as a whole, how the role assignment worked, how the work was distributed and if we encountered any difficulties related to language or culture seeing as the group consisted of two Norwegian students and two Serbian students.

13.4.1 Role assignment

As described in section 2.4 we assigned the group into the following roles:

- Scrum master: dynamic
- Communication responsible: Johan Reitan
- Development responsible: Dusan Stefanovic
- Documentation responsible: Johan Reitan
- Team leader: Dusan Stefanovic
- Test responsible: Jørgen Faret
- Quality assurance responsible: Nenad Milosavljevic
- Software repository responsible: Johan Reitan

- Secretary: Jørgen Faret

Having the scrum master role as a dynamic role worked well, because it gave every team member the opportuniy to lead a scrum cycle and therefore worked well in teaching every group member what being a scrum master entailed. Because we had a team leader in place for the development of the application, being the scrum master in the context of our project mainly entailed leading sprint pre- and post-planning meetings in addition to leading scrum stand-ups.
In general, the roles that had very clearly defined areas of responsibility were the roles that worked best during the project. This includes the roles communication responsible, documentation responsible, team leader, test responsible, software repository responsible and secretary. Seeing as the group worked as a quite cohesive unit, and was quite small, the roles quality assurance responsible and development responsible did not play as big of a part, and were maybe not as necessary as we had planned.

13.4.2 Work distribution

Initially, the plan was to divide the workload both in terms of coding and writing the report equally amongst the four group members, with the roles responsible for overseeing work on each of these two points (documentation responsible and development responsible) responsible for delegating specific tasks and ensuring progress in their respective areas.
During the course of the project, we gradually started dividing work so that Nenad and Dusan focused more on working on the development of the application, and Jørgen and Johan focused more on writing the report and testing the application. This was done because Nenad and Dusan were a little more familiar with developing iOS applications, while Jørgen and Johan were more the only team members familiar with writing in LaTeX and were slightly more comfortable with technical writing. After sprint one was concluded, and until the closing stages of the project, this remained the general distribution and functioned well.

13.4.3 Culture and language

Because our group was a group with two Norwegian and two Serbian students, ours was a group with a potential for language difficulties and misunderstandings due to different cultural backgrounds. This was something we as a group were aware of and discussed in one of our first meetings. Luckily, because all of the members of the group spoke english very well, we did not experience any difficulties due to language. We also all did a good job of making sure to be clear and concise when commicating to avoid the fact that we are from different cultures causing

any problems.
A result of our different nationalities, combined with the fact that the two group members from each country had quite similar areas of expertise, did result in the group members from each country working slightly closer with each other than with the group as a whole.

Bibliography

[1] Apple. *Conventions.* [Online; accessed 2013-10-14]. 2012. URL: https://developer.apple.com/library/ios/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/Conventions/Conventions.html.

[2] Apple. *Developer tools overview.* [Online; accessed 2013-10-14]. 2013. URL: https://developer.apple.com/technologies/tools/.

[3] Apple. *Quartz Core Framework Reference.* [Online; accessed 2013-10-11]. 2012. URL: https://developer.apple.com/library/mac/documentation/graphicsimaging/reference/QuartzCoreRefCollection/_index.html.

[4] William Bittle. *SAT (Separating Axis Theorem).* [Online; accessed 2013-11-08]. 2010. URL: http://www.codezealot.org/archives/55#sat-algo.

[5] Git. *Git.* [Online; accessed 2013-11-02]. 2013. URL: http://git-scm.com/.

[6] Philippe Kruchten. "Architectural Blueprints - The "4+1" View Model of Software Architecture". In: *IEEE Software* (Nov. 1995).

[7] Agile Manifesto. *Principles behind the Agile Manifesto.* [Online; accessed 2013-10-11]. 2001. URL: http://www.agilemanifesto.org/principles.html.

[8] Steve McConnell. *Software Quality at Top Speed.* [Online; accessed 2013-11-06]. 1996. URL: http://www.stevemcconnell.com/articles/art04.htm.

[9] Winston W. Royce. *Managing the development of large software systems.* [Online; accessed 2013-10-11]. 1970. URL: http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf.

[10] Ian Skerrett. *Eclipse Community Survey Results for 2013.* [Online; accessed 2013-10-07]. 2013. URL: http://ianskerrett.wordpress.com/2013/06/12/eclipse-community-survey-results-for-2013/.

[11] Wikipedia. *ISO/IEC 9126*. [Online; accessed 2013-11-15]. 2013. URL: `http://en.wikipedia.org/wiki/ISO/IEC_9126`.

Appendix A

Code conventions

A.1 Objective C

In order to ensure easier maintenance of the system, as well as creating a foundation for future development, we have agreed on some code conventions. All of these conventions comply to the rules defined by Apple [1].

Code blocks All code blocks that follow a function declaration, class declaration, conditional statement or control flow statement should start on a new line.

Indentation All code blocks must be intended. The starting brace can be on the same line as the statement or declaration it's enclosed by, or it can be on a new line.

Spacing after keywords After each keyword there should be a space. An `if` statement, for example, should be written as `if␣(condition)`, instead of `if(condition)`.

```
- (void)loadWalls:(TBXMLElement *)element toPlan:(NAVPlan *)plan;
```

Naming conventions We are going to follow the camelCase convention, which means that all phrases will be written without spaces, and the first letter in each word is capitalised except the first one. Function and variable names should be self explanatory. Table A.1 shows some examples.

Code examples

Table A.1: Naming conventions

Case	Rule	Example
Variables	lower camelCase	`anExampleVar`
Methods lower	camelCase	`myMethod()`
Classes Upper	camelCase	`TestClass`
Interfaces Upper	camelCase	`TestInterface`
Constants	all uppercase	`EXAMPLECONST`

```
//NAVPlanViewController.h

// the name of the variables starts with a small letter
@property (strong, nonatomic) NAVPlan *plan;
@property (strong, nonatomic) NSMutableArray *elements;
@property (strong, nonatomic) NSMutableArray *dimensions;

// function name begins with a lower case letter and is self
      explanatory
- (void)drawElements;

@end
```

```
// NAVPlanViewController.m

// define a constant, name is in capital letters
#define NUMBERS_ONLY @"1234567890."

//variables definition
@interface NAVPlanViewController () {
   CGFloat px;
   CGFloat py;
   CGPoint validCenter;
   bool positionInvalid;
   UIImageView *activeElement;
   UITextField *txtWidth;
}
```

```
// constructor method which includes an example of indentation and
       spacing after keywords
- (id)initWithCoder:(NSCoder *)decoder
{
    self = [super initWithCoder:decoder];
    if (self) {
        self.elements = [NSMutableArray array];
        self.dimensions = [NSMutableArray array];
        // Custom initialization
        NAVParser *parser = [[NAVParser alloc] init];
        self.plan = [parser loadPlanFromXML:[[NSBundle mainBundle]
               pathForResource:@"testPlan" ofType:@"xml"]];
    }
    return self;
}
```

A.2 XML

Our XML (read about XML in 3.2.1) follows a simple structure: every plan element is assigned its own tag which contains all the required attributes for that specific element. For example, walls are defined in a tag named `<wall>`, and have start and end points, with tags respectively named `<startPoint>` and `<endPoint>`. These, in turn, have x and y coordinates as attributes. The document is structured so that it is easily read and interpreted by humans, mostly for the sake of control while developing.
An example plan in XML:

```
<?xml version="1.0"?>
<plan>
    <name>Plan name</name>
    <description>Plan description</description>
    <wall>
        <startPoint>
            <x>100</x>
            <y>600</y>
        </startPoint>
        <endPoint>
            <x>100</x>
            <y>100</y>
        </endPoint>
    </wall>
    <rectangular>
        <originPoint>
            <x>400</x>
            <y>350</y>
        </originPoint>
        <name>Bed</name>
        <height>128</>
        <width>202</>
        <icon>images</>
        <angle>130</angle>
    </rectangular>
</plan>
```

Appendix B

User manual

Contents

B.1 Creating a plan

How do I create a plan?

1. Tap the "Ny plan" button on the menu screen
2. Enter the information to describe the plan
3. Tap the "Lag plan" button

B.2 Removing a plan

How do I remove a plan?

1. Tap the "Åpne plan" button on the menu screen

2. From the list of plans, select the one to be deleted
3. Tap the button "Slett plan"

B.3 Loading a plan

How do I load a plan?

1. Tap the "Åpne plan" button on the menu screen
2. From the list of plans, select the one to open
3. Tap the "Åpne" button

B.4 Editing a plan

B.4.1 Adding elements

How do I open the menu for adding elements to a plan?

1. On the screen displaying the plan, tap the "Legg til" button in the lower left corner
2. A list of elements will appear on the left side of the screen
3. Tap the "Legg til" button again to close the list

How do I add walls to a plan?

1. Choose "Vegger" in the list of elements, which makes the wall submenu appear
2. Select between the different wall elements:
   (a) If "Frihånd" is selected, tap and drag across the plan to draw a wall of desired length
   (b) If one af the fixed walls is selected, tap on the screen where you want the wall to start, and then enter the length of the wall
3. Save elements to the elements list for future use by tapping the "Lagre" button
4. For using saved, custom walls, select one from the list and tap the screen at the starting point

How do I add items to a plan?

1. Choose the "Elementer" submenu in the list
2. Select the item to add
3. Tap the screen to choose the position of the element
4. A screen for entering measurements will show up
5. Save elements to the elements list for future use by tapping the "Lagre" button
6. Insert an element by tapping the "Legg til" button

How do I add a room to the plan

1. Choose the "Rom" submenu in the list
2. Select between rooms. "Med klokken" eller "Mot klokken" means the direction that the room shall be drawn starting from the tapped point.
3. Tap the screen to choose the start point for the room
4. A screen for entering the measurements will appear
5. Save elements to the elements list for future use by tapping the "Lagre" button
6. insert the room by tapping the "Legg til" button

B.4.2 Transforming elements

How do I resize an item?
There are two ways to resize items:

1. (a) Click on the element
   (b) A resize/rotate menu will appear. Insert the measurements and tap the "Endre" button
2. (a) Pinch items with two fingers on the screen. Both touch points must be on the item

How do I rotate items?
There are two ways to rotate items:

1. (a) Click on the element
   (b) A resize/rotate menu will appear. Insert the angle and tap the "Endre" button
2. (a) Touch the item with two fingers
   (b) Start rotating both fingers around the center of the element

How do I rotate walls?
If you want to specify the angle between two walls manually, do the following:

1. Long press the connecting corner
2. A rotate menu will appear. Insert the angle and tap one of the buttons "Blå" or "Gul" to choose which wall to rotate.
3. A confirmation dialog will appear. If the angle is not correct, tap the "Motsatt vinkel" button. Otherwise, tap the "OK" button.

How do I move items?

1. Tap the item an hold onto it while dragging it around
2. The item will turn red if there are obstacles (other items or walls) at its current position. If the item is released when it is red, it will be placed back to the last valid position.

How do I move walls?

1. Tap one of the wall's corners, and hold onto it while dragging it around.
2. The wall will turn red if there are obstacles (other items or walls) at its current position. If the item is released when it is red, it will be placed back to the last valid position.

How do I activate special functions for an item?

1. A long tap on the item will set its state to active. What happens depends on the type of item (e.g. for a wheelchair, all doors which are too narrow for the wheelchair to pass through will turn red, and a radius showing the wheelchair rotation will be shown.
2. Long tap the item again to set its state to inactive.

How do I zoom in and out in the plan?
This can be done in two ways:

1. By tapping the "Zoom inn" and "Zoom ut" buttons at the bottom of the plan view screen
2. By pinching the plan view with two fingers

How do I save a plan?

1. Tap the "Lagre" button on the bottom of the plan view screen

B.5 Customize UI

How do I hide dimensions?

1. Tap the "Skjul mål" button at the bottom of the plan view screen
2. Tapping again will show the dimensions

How do I hide corners?

1. Tap the "Skjul hjørner" button at the bottom of the plan view screen
2. Tapping again will show the corners

How do I lock scrolling and zooming of the plan?

1. Tap the "Lås zoom"/"Lås panorering" buttons in the bottom right of plan view screen.
2. Tap again to unlock zoom/scroll.

B.6 Exporting and sharing a plan

How do I export a plan?

1. Tap the "Eksporter" button at the bottom of the plan view screen
2. Choose the format for exporting the plan. The exported file is saved on the device.

How do I share a plan?

1. Tap the "Del" button at the bottom of the plan view screen
2. Choose e-mail in the sharing dialog
3. Choose format for sharing
4. Fill out the necessary fields to send the e-mail

Appendix C

Test executions

C.1 Sprint 1 integration tests

Item	*Description*
Name	Navigate windows
Identifier	INTG-11
Features to be tested	The possibility to navigate through the different windows and correctness of program flow
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Main menu is open
Execution steps	1. User presses 'Ny plan' 2. User presses 'Tilbake' 3. User presses 'Åpne plan' 4. User presses 'Tilbake'
Success criteria	Application must properly navigate between pages
Test result	Success
Test responsible	Jørgen Faret

Table C.1: Test case INTG-11

Item	*Description*
Name	Load plan
Identifier	INTG-12
Features to be tested	Opening a plan
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Main menu is open 2. Plan is created
Execution steps	1. User presses 'Åpne plan' 2. User selects a plan from the list of plans 3. User presses 'Åpne'
Success criteria	The correct plan is opened and properly displayed
Test result	Success
Test responsible	Jørgen Faret

Table C.2: Test case INTG-12

Item	*Description*
Name	Create item
Identifier	INTG-13
Features to be tested	Adding an element to the plan
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses 'Legg til' 2. User selects an item to add and drags it to plan
Success criteria	Item is displayed in plan
Test result	Success
Test responsible	Jørgen Faret

Table C.3: Test case INTG-13

C.2 Sprint 2 integration tests

Item	*Description*
Name	Create item
Identifier	INTG-21
Features to be tested	Moving an element in the plan
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. An element is created
Execution steps	1. User selects an element by pressing it 2. User drags the element to the desired location
Success criteria	Element is moved to the correct location
Test result	Success
Test responsible	Jørgen Faret

Table C.4: Test case INTG-21

Item	*Description*
Name	Create item
Identifier	INTG-22
Features to be tested	Checking for intersecting points between two items
Priority	Medium
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. Two items are created
Execution steps	1. User selects an item by pressing it 2. User drags the item so that it is overlapping with another item
Success criteria	Item becomes red and upon release of element it moves back to original location
Test result	Success
Test responsible	Jørgen Faret

Table C.5: Test case INTG-22

Item	*Description*
Name	Create item
Identifier	INTG-23
Features to be tested	Checking for intersecting points between wall and item
Priority	Medium
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. An item is created 4. A wall is created
Execution steps	1. User selects an item by pressing it 2. User drags the item so that it is overlapping with a wall
Success criteria	Item becomes red and upon release of element it moves back to original location
Test result	Failure
Test responsible	Jørgen Faret

Table C.6: Test case INTG-23

Item	*Description*
Name	Create item
Identifier	INTG-24
Features to be tested	Resizing an item
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. An item is created
Execution steps	1. User selects an item by pressing it 2. User selects two points on top of the item and moves the points either towards or away from the center of the item.
Success criteria	Item increases in size if points are moved towards center of element and decreases in size if points are moved away from center of element
Test result	Success
Test responsible	Jørgen Faret

Table C.7: Test case INTG-24

Item	*Description*
Name	Create item
Identifier	INTG-25
Features to be tested	Rotating an item
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. An item is created
Execution steps	1. User selects an item by pressing it 2. User selects two points on top of the item and moves the points in the same direction around the center of the item
Success criteria	Item is rotated with the direction of the movement of the points
Test result	Success
Test responsible	Jørgen Faret

Table C.8: Test case INTG-25

Item	*Description*
Name	Create item
Identifier	INTG-26
Features to be tested	Removing an item
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. An item is created
Execution steps	1. User selects an item by pressing and holding for a short amount of time 2. User presses delete on the popup menu
Success criteria	Item is removed from the plan
Test result	Success
Test responsible	Jørgen Faret

Table C.9: Test case INTG-26

C.3 Sprint 3 integration tests

Item	*Description*
Name	Create item
Identifier	INTG-31
Features to be tested	Navigating menu for adding items
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses 'Legg til' 2. User presses 'Vegg' in the list of element categories 3. User presses 'Tilbake' 4. User presses 'Gjenstand' in the list of element categories 5. User presses 'Tilbake'
Success criteria	Application properly navigates in the menu
Test result	Success
Test responsible	Jørgen Faret

Table C.10: Test case INTG-31

Item	*Description*
Name	Create item
Identifier	INTG-32
Features to be tested	Adding walls
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses ’Legg til’ 2. User presses ’Vegg’ in the list of elements 3. User presses a point in the plan 4. User drags to a desired endpoint for the wall
Success criteria	A wall is created starting at the startpoint and ending at the endpoint
Test result	Success
Test responsible	Jørgen Faret

Table C.11: Test case INTG-32

Item	*Description*
Name	Create item
Identifier	INTG-33
Features to be tested	Adding doors
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. A wall is created
Execution steps	1. User presses 'Legg til' 2. User selects 'Gjenstand' in the list of element categories 3. User presses 'Dør' 4. User presses a point in the plan 5. User drags the door and attaches in on top of a wall
Success criteria	Door is displayed properly on top of a wall
Test result	Success
Test responsible	Jørgen Faret

Table C.12: Test case INTG-33

Item	*Description*
Name	Create item
Identifier	INTG-34
Features to be tested	Adding items
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses 'Legg til' 2. User selects 'Gjenstand' in the list of element categories 3. User presses a point in the plan not conflicting with other elements to put the item
Success criteria	Item is properly displayed at the correct location.
Test result	Success
Test responsible	Jørgen Faret

Table C.13: Test case INTG-34

Item	*Description*
Name	Create item
Identifier	INTG-35
Features to be tested	Saving and loading a plan
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan with a set of elements is created 2. Application is in plan viewer
Execution steps	1. User presses 'Lagre' 2. User exits the plan viewer 3. User presses 'Åpne plan' 4. User finds the previous plan in the list of plans 5. User opens the plan
Success criteria	Plan is displayed with the same elements in the location with the same size as when it was saved
Test result	Success
Test responsible	Jørgen Faret

Table C.14: Test case INTG-35

Item	*Description*
Name	Create item
Identifier	INTG-36
Features to be tested	Plan metadata properly saved
Priority	Low
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Main menu is open
Execution steps	1. User presses ’Ny plan’ 2. User inputs desired plan metadata and presses ’Lag plan’ 3. User presses ’Lagre’ 4. User presses ’Tilbake’ 5. User presses ’Åpne plan’ 6. User finds the previous plan in the list of plans and inspects metadata
Success criteria	Plan is displayed with correct metadata
Test result	Success
Test responsible	Jørgen Faret

Table C.15: Test case INTG-36

Item	*Description*
Name	Create item
Identifier	INTG-37
Features to be tested	Zooming in the plan
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User selects two points in the plan where there are no elements and drags the points further away or closer to each other
Success criteria	The plan zooms out if the points were dragged further from each other and zooms in if they were dragged further away from each other
Test result	Success
Test responsible	Jørgen Faret

Table C.16: Test case INTG-37

Item	*Description*
Name	Create item
Identifier	INTG-38
Features to be tested	Panning in the plan
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User selects a point in the plan where there are no elements and drags the point
Success criteria	Plan is panned in the same direction as the point is dragged
Test result	Success
Test responsible	Jørgen Faret

Table C.17: Test case INTG-38

C.4 Sprint 4 integration tests

Item	*Description*
Name	Create item
Identifier	INTG-41
Features to be tested	Creating custom elements
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses 'Legg til' 2. User selects 'Gjenstand' in the list of element categories 3. User selects 'Brukerdefinert gjenstand' 4. User presses a point in the plan not conflicting with other elements to put the item 5. User enters the desired height and width and label of the item
Success criteria	Item is properly displayed at the correct location with the correct size.
Test result	Success
Test responsible	Jørgen Faret

Table C.18: Test case INTG-41

Item	*Description*
Name	Create item
Identifier	INTG-42
Features to be tested	Using saved custom elements
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses 'Legg til' 2. User selects 'Gjenstand' in the list of element categories 3. User selects the desired custom element 4. User presses a point in the plan not conflicting with other elements to put the item
Success criteria	Item is properly displayed at the correct location with the correct size.
Test result	Success
Test responsible	Jørgen Faret

Table C.19: Test case INTG-42

Item	*Description*
Name	Create item
Identifier	INTG-43
Features to be tested	Copying elements
Priority	Medium
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. An element is created
Execution steps	1. User presses an element 2. User presses 'Kopier' in the pop-up menu 3. User presses a point in the plan not conflicting with other elements to put the item
Success criteria	Item is properly displayed at the correct location.
Test result	Success
Test responsible	Jørgen Faret

Table C.20: Test case INTG-43

Item	*Description*
Name	Create item
Identifier	INTG-44
Features to be tested	Creating horizontal/diagonal walls
Priority	Medium
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses ’Legg til’ 2. User selects ’Vegg’ in the list of element categories 3. User selects ’Horisontal vegg’ 4. User presses a point in the plan not conflicting with other elements to draw the wall 5. User sets wall length 6. User selects ’Vertikal vegg’ 7. User presses a point in the plan not conflicting with other elements to draw the wall 8. User sets wall length
Success criteria	Horizontal and vertical walls are displayed properly
Test result	Success
Test responsible	Jørgen Faret

Table C.21: Test case INTG-44

Item	*Description*
Name	Create item
Identifier	INTG-45
Features to be tested	Creating rooms
Priority	Medium
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses 'Legg til' 2. User selects 'Rom' in the list of element categories 3. User inputs room dimensions 4. User presses a point in the plan not conflicting with other elements to draw the room
Success criteria	Room is properly displayed at the correct location.
Test result	Success
Test responsible	Jørgen Faret

Table C.22: Test case INTG-45

Item	*Description*
Name	Create item
Identifier	INTG-46
Features to be tested	Exporting plans
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. Plan with elements is created
Execution steps	1. User presses 'Eksporter' 2. User chooses a file format for eksporting
Success criteria	A correct picture of the plan is exported.
Test result	Success
Test responsible	Jørgen Faret

Table C.23: Test case INTG-46

Item	*Description*
Name	Create item
Identifier	INTG-47
Features to be tested	Sharing plans
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. Plan with elements is created
Execution steps	1. User presses 'Del' 2. User selects a plan as file format for sharing 3. User inputs the e-mail desired recipient and meta-data for the e-mail 4. User presses send 5. User opens the e-mail inbox of the recipient 6. User opens the plan attachment in the e-mail
Success criteria	Plan is opened and is unchanged from when it was sent
Test result	Failure
Test responsible	Jørgen Faret

Table C.24: Test case INTG-47

Item	*Description*
Name	Create item
Identifier	INTG-48
Features to be tested	Deleting plans
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Main menu is open
Execution steps	1. User presses 'Åpne plan' 2. User selects a plan from the list of plans 3. User presses 'Slett'
Success criteria	Plan is removed from list of plans and the plan's xml-file is deleted from the folder containing plans
Test result	Failure
Test responsible	Jørgen Faret

Table C.25: Test case INTG-48

Item	*Description*
Name	Create item
Identifier	INTG-49
Features to be tested	Locking the plan zoom and pan
Priority	Medium
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses 'Lås skjerm' 2. User attemps to pan plan by dragging a point in the plan 3. User attemps to zoom plan by selecting two points in the plan and dragging them further away or closer to each other
Success criteria	The plan viewer does not pan or zoom
Test result	Success
Test responsible	Jørgen Faret

Table C.26: Test case INTG-49

Item	*Description*
Name	Create item
Identifier	INTG-410
Features to be tested	Zooming using buttons
Priority	Medium
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses 'Zoom ut' 2. User presses 'Zoom inn'
Success criteria	Plan zooms in when user presses 'Zoom inn' and zooms out when user presses 'Zoom ut'
Test result	Success
Test responsible	Jørgen Faret

Table C.27: Test case INTG-410

Item	*Description*
Name	Create item
Identifier	INTG-411
Features to be tested	Zooming using buttons
Priority	High
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer
Execution steps	1. User presses 'Legg til' 2. User selects 'Gjenstand' in the list of element categories 3. User selects 'Rullestol' 4. User inputs the desired dimensions of the wheelchair in the popup menu 5. User presses a point in the plan not conflicting with other elements to place the wheelchair
Success criteria	A wheelchair element is placed at the desired location
Test result	Success
Test responsible	Jørgen Faret

Table C.28: Test case INTG-411

Item	*Description*
Name	Create item
Identifier	INTG-410
Features to be tested	Zooming using buttons
Priority	Medium
Testing technique	Integration testing
Testing method	Gray-box testing
Pre-conditions	1. Plan is created 2. Application is in plan viewer 3. At least one wall is created 4. At least one door is created 5. A wheelchair with a diameter larger than the width of the door is created
Execution steps	1. User presses the wheelchair and holds until the wheelchair border is green
Success criteria	The door created becomes red to illustrate that the wheelchair won't fit through it
Test result	Success
Test responsible	Jørgen Faret

Table C.29: Test case INTG-412

C.5 System tests

Item	*Description*
Name	Create plan
Identifier	SYST-1
Testing technique	Black box
Requirements to be tested	• **NFR1** • **NFR3** • **NFR4** • **FR1** • **FR3**
Pre-conditions	None
Execution steps	1. User opens application 2. User creates a new plan, making sure to fill in all of the metadata 3. Users saves plan 4. User closes application
Success criteria	1. All non-functional requirements are fulfilled 2. Program flow is correct 3. All buttons pressed produce expected response 4. No unexpected program behavior is observed
Test result	Success
Test responsible	Jørgen Faret

Table C.30: Test case SYST-1

Item	*Description*
Name	Populate plan
Identifier	SYST-2
Testing technique	Black box
Requirements to be tested	• **NFR1** • **NFR3** • **NFR4** • **FR2** • **FR3** • **FR4** • **FR5** • **FR7** • **FR11** • **FR12** • **FR14** • **FR15**
Pre-conditions	An empty plan is created
Execution steps	1. User opens application 2. User loads a created plan 3. User populates the plan adding at least one item of each category 4. User saves plan 5. User closes application
Success criteria	1. All non-functional requirements are fulfilled 2. Program flow is correct 3. All buttons pressed produce expected response 4. No unexpected program behavior is observed
Test result	Success
Test responsible	Jørgen Faret

Table C.31: Test case SYST-2

Item	*Description*
Name	Edit plan
Identifier	SYST-3
Testing technique	Black box
Requirements to be tested	• **NFR1** • **NFR3** • **NFR4** • **FR2** • **FR3** • **FR7** • **FR8** • **FR9** • **FR10** • **FR11** • **FR12** • **FR14** • **FR15**
Pre-conditions	A plan that is populated with elements is created
Execution steps	1. User opens application 2. User loads created plan 3. User rearranges the elements in the plan to create a realistic room with items, making sure to perform the actions move, rotate, resize and remove at least once 4. User saves plan 5. User closes application
Success criteria	1. All non-functional requirements are fulfilled 2. Program flow is correct 3. All buttons pressed produce expected response 4. No unexpected program behavior is observed
Test result	Success
Test responsible	Jørgen Faret

Table C.32: Test case SYST-3

Item	*Description*
Name	Export and delete plan
Identifier	SYST-4
Testing technique	Black box
Requirements to be tested	• **NFR1** • **NFR3** • **NFR4** • **FR2** • **FR16**
Pre-conditions	A plan that is populated with elements is created
Execution steps	1. User opens application 2. User loads created plan 3. User exports plan via e-mail 4. Users deletes plan 5. User closes application 6. User opens application 7. User verifies that plan is deleted
Success criteria	1. All non-functional requirements are fulfilled 2. Program flow is correct 3. All buttons pressed produce expected response 4. No unexpected program behavior is observed
Test result	Success
Test responsible	Jørgen Faret

Table C.33: Test case SYST-4