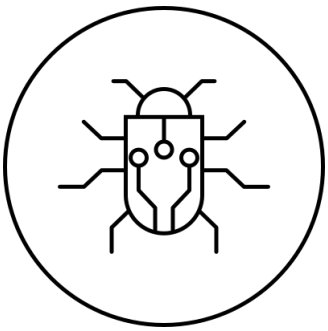
Danmarks Tekniske Universitet

02160 Agile Software Development:
Hospital Information Management Suite

8. May 2019

Mojn X2000 - heisenbug



Authors:	***Study No:***
Andreas Heidelbach Engly	s170303
Anton Ruby Larsen	s174356
Karl Emil Takeuchi-Storm	s130377
Mads Esben Hansen	s174434
Mathias Jarl Jarby Søndergaard	s174426

Contents

Introduction

The following paper describes the process of developing a Hospital Information Management Suite (HIMS) for use in Tanzanian hospitals and medical clinics. The concrete system is named Mojn X2000, and the development team heisenbug.

The management system is designed to improve the efficiency of the working process on the hospital in Tanzania. The workday is often stressful due to high temperatures, and poor financial conditions. It is supposed to help the staff systematize the documentation of the patients, and centralize key information. It is also supposed to help keep track of which patients are admitted to which departments, and likewise with medical staff. It is a mean to minimize administrative workload, thus the goal is to use as little time interaction with the system as possible.

1.1 Evolution Processes

The project has been a learning experience in management, agile principles as well as software development. As such the team has gone through a series of development steps, which followed the steps in both software development and agile structure.

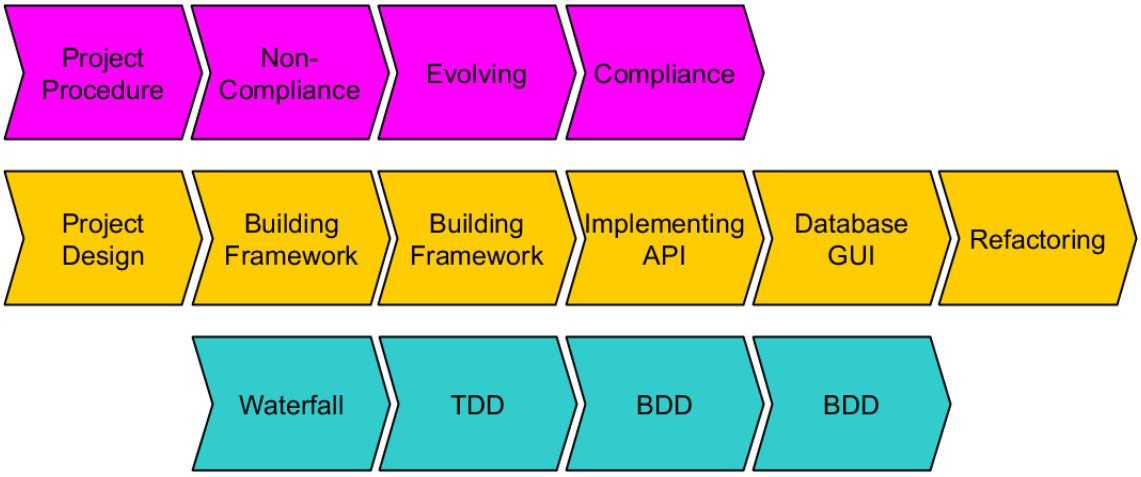


Figure 1.1: Correlation between Agile, Software and Team evolution

The above figure shows these concurrent developments, and are each explained in more detail in their respective sections below.

1.2 Software Structure

To demonstrate the software a short demonstration video can be found on the below link.
`https://www.youtube.com/watch?v=AZ-29yj2x1E`

In order to design a system to manage hospitals, the system must support several concurrent users, which presented the team with a set of choices and challenges. Further the system had to be expandable and adaptable. The current setup is made as a stand alone structure, where the database is running online, with the GUI calling an API layer.

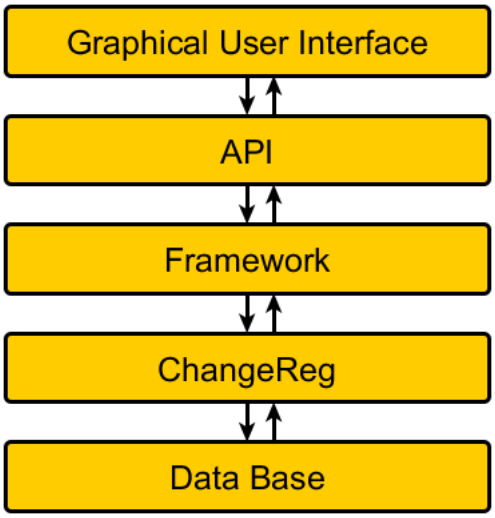


Figure 1.2: Overview of the implemented program structure

This setup enables the software to easily detach the database, to run as a true server-less setup, where a lot of clients can call the database. Expanding the usage further is possible in the form of a webserver setup, where a new client, would be a server running a website, calling the database, and a json-adapter is implemented to support a web-UI.

Extending on the choices, a few interpretations were made towards implementing the project, these would normally be discussed with a client, but in the project the client is fictive, thus the following implementations were decided upon. The emergency and out-patient departments is implemented as just an outpatient department, which contains a priority queue, where patients are either assigned a triage level or automatically receive lowest priority. Further bed assignment is automated, such that a patient is assigned to an available bed upon admission, and if the department wants to assign a different bed, they can move the patient instead of the double work implementation, where the patient is first admitted and subsequently assigned to a bed. Lastly a change-log was added to the system, to enable a system admin to check who did what when. Further a change-log rebuild tool would add additional persistency to the deployment of the software.

Project Management

The initial phase of the project was to create a plan for how to progress, rather than using rigid project planning, similar to a construction site, we made an overall loose plan of when to implement what main parts. This should make it possible for the team to use an agile approach, focused around sprints using a Kanban board for back-log items.

As such the initial project plan main focus was on how to implement the development process (See appendix C), select tools and setup procedures (See appendix D).

As describe above the main focus was to implement an agile process revolving around sprints, however throughout the process both LEAN and JIT principles has been a strong influence of the development process. On each retro-perspective meeting we have discussed how to proceed, without creating waste and keep pushing forward taking the teams abilities into account. The project plan has evolved through these meeting, and is now significantly different than the initial version. The team has continuously evaluated the situation, and decided the best approach forward. In retro-perspective, the team's decisions has not always been perfect, but at each meeting, it was clear that the solution seemed most feasible at the time and skill level the team was on.

LEAN and JIT

The main thought being LEAN is to be efficient and evolving, this is done by carrying some base ideas in the processes and behavior. These principles has been followed in the project, but not as a rigid procedure, but rather as the underlying thought set. One exception is the Defer Decision principle, where we have not developed several options for implementations, but rather refactored to a better option where it made sense. Under this principle is also our strive to keep the code expandable and adaptable. In JIT the main focus is to do things when they are necessary to deliver and not before, this was used to aid the team in selecting back-log items. This helped us ensure that, code did not become obsolete before it was implemented and connected to the remaining code base.

2.1 Tooling

The tooling chosen for this project is described in our "Best Practice", where the usage and advantages are described (See appendix D). At an early stage in the project the tooling used, proved more a challenge, than an aid, especially regarding the use of git and github. To overcome these challenges the team appointed a "git master" in charge of all merging and aiding the team towards compliance to the correct usage. The "git master" deployed another software tool, Gitkraken which is not described in the tooling section, as it is a sub tool to git. The Gitkraken is an impressive visual tool to handle git and provides a good overview. Another challenge raised, when the team decided to implement Maven build automation, which on several occasions caused corruption in various files. It seemed common that the dependencies get corrupted and on one occasion the .classpath file was corrupted, and many man-hours were spend before the project would compile correctly again. The learning curve of Maven as intended is steep in the first place, moreover the bugs and work-arounds necessary to deploy the tool in practice made the usage a poor choice. With the current knowledge of Maven, deploying the tool seems feasible, however the tool has been hampering the project on an overall level. The advantage is that the team has developed and grown, and are now able to deploy the tool in the future, something that is a goal in itself thinking about the LEAN principle "Create Knowledge".

2.2 Procedures

As the project proceeded it became clear, that a firm test strategy was necessary to aid the compliance of the team, consequently an BDD procedure were implemented (See appendix D). The idea was to continue

implementing further procedure, however as the development was the main focus and the team rather small, no further procedures were implemented. If the project were to evolve further with the same team, this area would have gained more focus. The main advantage of a procedure is that discussions of interpretations is lessened and any updates to the procedure is decided by the entire team. This enables the team to work in a uniform and documented manner, allowing degrees of freedom only in the direction, where this is improving the production, but limiting the ones that hampers it.

2.3 User Manual

The full usage of the program "Mojn X2000" can be found in the User Manual under appendix B, which contains detailed description of how to perform simple operations within the system. The user manual is written with the mindset that the program should be as easy to operate as possible, therefore the manual is written in an easily readable language with brief description of each operation.
All descriptions are written in a bullet point manner, which are all easy to follow. This ensures that non experts in IT should be able to use the system optimally solely from the User Manual.
Furthermore the User Manual is written such that it can be used as a check list whenever a hospital employee needs to use the IT system. If it is followed, no error should occur, and everything should work as described.

Agile Development

Agile development is essential for creating big scale software without major time losses. Furthermore, it is very fruitful for avoiding conflicts code wise, but also management wise. This section covers our adventure into the Agile world, where we describe our evolving work flow. We comment on our process iterations and different development types.

3.1 Process Evolution

As described in the Project Plan, the management process is to be reviewed after each sprint. It turned out to be a very important aspect of the project, as the initial aim were set too high taking the team's skill-set into account. However the team managed to recover and change the process to fit the requirements and initial goals, as the team skills improved.

First Process Iteration

The first sprint process contained both Test Driven Development (TDD) and Behavior Driven Development (BDD), which turned out to be a process that was hard to adhere to, in retro-perspective a failure to understand the difference between the presented agile processes.

The first sprints succeeded in the way a lot of code were developed, however the team failed to follow procedures correctly. Thus on the first Retro-Perspective meeting, this issue were thoroughly discussed and rather than changing the process, it was decided to adhere to the procedure, however it turned out that the confidence in the use of tools and lacking programming experience, made this impossible.

Again the result of coding was good, and the project as pure coding were on rails, however BDD was not implemented at all, and TDD were lacking. Before calling the sprint to be ended, the testing scheme for TDD was updated and coverage were good.

Second Process Iteration

On the following (second) Retro-Perspective meeting, the team decided to update the sprint process, realizing that the abilities in TDD were in place to implement a proper TDD process, however not so for a BDD process. Consequently a call to proceed with a pure TDD sprint setup were made, this to avoid generating waste and not progressing in the following sprints. A clear definition of how the team should implement TDD were also agreed upon, and added to the "Best practice" handbook.

This process were in use for additional three sprints, and these were highly productive. The foundation for the project were nearly in place.

Third Process Iteration

In order to fulfill the User Story requirement and the fact that the development had reached a state, where implementing an API made sense, the team decided to build the API on a BDD based sprint structure.

As a result the team had to adapt to a new sprint cycle process again, and subsequently two sprint failed completely, with no positive outcome in terms of coding produced. However, seen from a learning perspective, these two sprints resulted in the whole team, getting very good at BDD, and finally the use of git was thoroughly understood as well.

Thus on the seventh Retro-Perspective meeting the development team, decided to completely overhaul the toolkit used for development so far, and tweak the process yet again. Finally a good BDD-process were in place, without generating waste from rewriting code that were already implemented.

3.2 Test Driven Development

Test Driven Development, as the name suggests, is a process by which the tests are written before the code. This doesn't mean writing all the tests and then writing all the code. It is subtler than that. You start by writing just a very small test. Then you write just enough code to get the test to pass. Then you write another small test and then it is just rinse and repeat. The code and the tests are born and grow together and hereof the test-ability of the code is "built in". The key to a unit test is to test the "unit" in isolation. If you do not comply to this the whole strength of TDD disappears because your ability to identify the specific unit that does not work fails. An example of this could be if you had written a calendar that supports leap years and you where feeding the machine more than just leap years but whole persons where some had birthday in a leap year and some not. If the system failed, you would not be able to tell if it was the functions fault or if it had something to do with another feature dealing with the person.

3.3 Behavior Driven Development

The first thing to notice about Behavior Driven Development is that the test is written in - more or less - plain English. You write them as what is called a user story. These will be described in detail further down but for know they just consist of three parts. A Context part, a Event part and a Outcome part. An example could be "Given my bike is fully functional, When I wake up in the morning, Then I can bike to school". This is a very direct way to describe the behaviours that the customer can expect from the system and a set of behaviours will together form a contract. The plain English nature of the "Context, Event, Outcomes" format means that the business people of a company easily can describe the behaviours that matter to the customer. Another benefit is that Behavioural Tests helps the developers to only write whats necessary and not allot of unused code that the customer does not pay for. One of the drawbacks is that when a test fail you do not know why. You just know that something is wrong.

User Stories

A user story describes a specific usage feature, which is concise and independent. The stories will create value in that they will be turned into a set of acceptance criteria, which again will become test scenarios. Each story contains an easily identifiable user, a feature described in a way that both a coder and a costumer can understand it and a goal, which the given user wishes to achieve utilizing the feature.

After the user stories have been developed and agreed upon, acceptance criteria for each is created. Each criterion will be used to produce a scenario corresponding to said criterion. These scenarios will then be implemented as BDD tests, which should be agreed upon to be used as acceptance tests with the customer. Then software is developed to pass all tests and handed to the customer.

One of the benefits of creating User Stories, which for the new students of Agile development may seem superfluous, is the customer to programmer relation and work flow. Having User stories as a part of the development process, unlocks the ability for the customer to have a greater understanding of exactly what is being developed. In addition, it ensures that the program being developed has the right functionalities and no dead or unneeded code is created. Furthermore, the user stories we created provided a solid framework for our later API implementation, which was a much needed structural pillar.

Implementation of User Stories

We controlled the creation of User stories using trello, where we defined a set workflow for designing the user stories. We choose our trello setup following the kanban style, giving us an excellent overview. To be specific we used the order: Back-log ⇒ Define Scenario and Cucumber Test ⇒ Implement ⇒ Test ⇒ Document ⇒ Done

However, how did we define and specify what user stories was needed? After studying the project description, we split different wanted user stories into categories having the title of what assignment they belonged to. This is shown in the following figure.

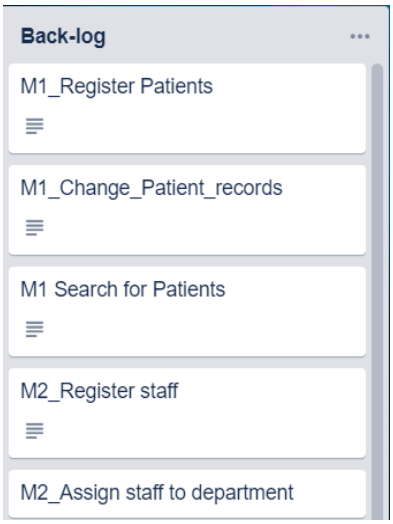


Figure 3.1: Snippet of the back-log and some user Stories.

After using this method to map what user stories we needed to fulfill the project's description, we ended up having about 19 items in our back-log. These were the base of the various sprints to come.

Following our work flow, we now had to define the scenarios for our back-log items. This was done using Cucumber, which implements Gherkin. Gherkin is a powerful tool to write easy readable and understandable tests. An example of one of these cucumber tests is Change Patient info:

```
@tag
Feature: Change Patient Info

  Background:
  Given I have a patientID of a patient and I want to change their personal info

  @tag1
  Scenario: Succesful Change
    When I am entering a valid patientID
    And I am changing the given info to something valid
    Then I get a message that the change was succesful
  @tag2
  Scenario: Invalid patientID
    When I am entering invalid patientID
    Then I get a message that the patient does not exist
  @tag3
  Scenario: Illegal info change
    When I am entering a valid patientID
    And I am changing the info to something illegal
    Then I get a message that information entered was illegal
```

Figure 3.2: Cucumber test for change patient info.

Furthermore, figure 3.2, shows the structure of a Gherkin file. The feature being implemented is "Change Patient Info" which has a given Background corresponding to the user story. Then different scenarios for the specific feature and background is created. A successful change scenario is a must, and then some error scenarios are also provided, in case of faulty inputs.

Our thought process behind the creation of the user stories tried to follow the "Characteristics of good user stories" practice. We aimed for having the following 6 properties: Independent, Negotiable, Valuable, Estimable, Small and Testable. Of these 6 properties we primarily focused on valuable and small, as focusing on these enabled us to work the fastest.

3.4 Test Coverage

We aimed for achieving high coverage of the important methods and classes in our project. The important classes is the API and most of the framework, as these could be tested using JUnit. Other classes we had trouble testing, like most of the GUI classes. However, the GUI is tested every time one uses the program and errors are picked up here. Thereby, the GUI is "user-tested".
Finally, it was hard to create testing for the GUI without creating a specific tool. Perspective wise, we would create a Click-Bot driven by image recognition and some form of basic logic which would be able to use the GUI and test all functionalities. But creating such a tool is extremely out of the scope and would probably end up in a huge waste of time. Another approach we could take in order to test the GUI, is using libraries like SwingUnit and JFCunit, however we deemed this out of the scope aswell.

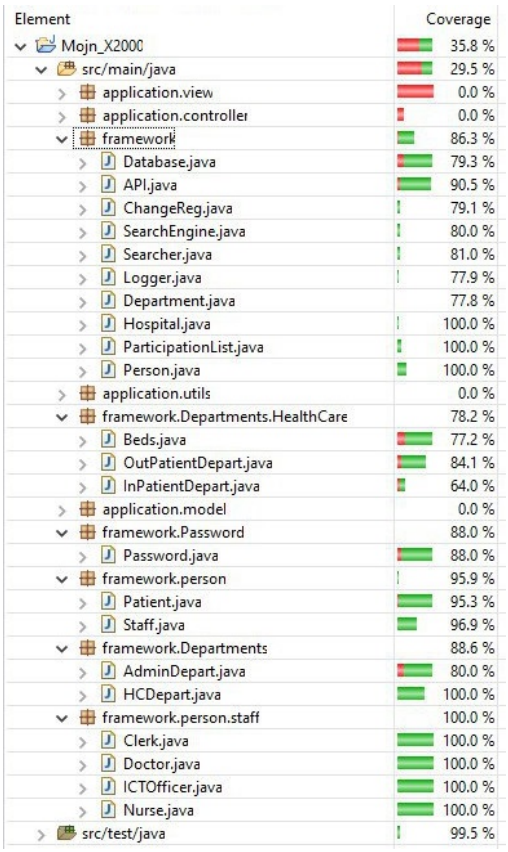


Figure 3.3: JUnit coverage of the project.

As seen from Figure 3.3, the whole project has a weak 36% coverage. However, this is mostly due to the GUI not being covered. The API has a whopping 91% coverage. Due to some error statements not being entered, the API is not closer to 100% - however the core methods are covered.

The whole framework class has around 86% coverage. Ideally it should be close to 100%, but due to the Database class in the framework package, this is not so. The database contains some error statements that are not entered doing testing. These error statements are just an extra layer of security, and the need to test them is not present. They are basically try catch, with error prints.

Program Structure

Our program can generally be divided into two parts. The graphical user interface and the back-end. The back-end can again be separated into three parts, Person structure, Hospital structure and the functional classes behind the hospital that deals with all actions done at the hospital. These different parts are implemented using different techniques and structures and those will be explained in the following section.

4.1 UML

Here we have shown three very simplified UML diagrams that only tell something about the structure of the code. Full UMLs with all public methods and attributes can be found in appendix E. Figure 4.1 shows how we have interpreted persons in our hospital. Here we have an abstract super class Person and then different types of persons extends this class as abstract Staff and Patient does here. A further extension is then made of the abstract class Staff where the different types of staff members each has one class each.

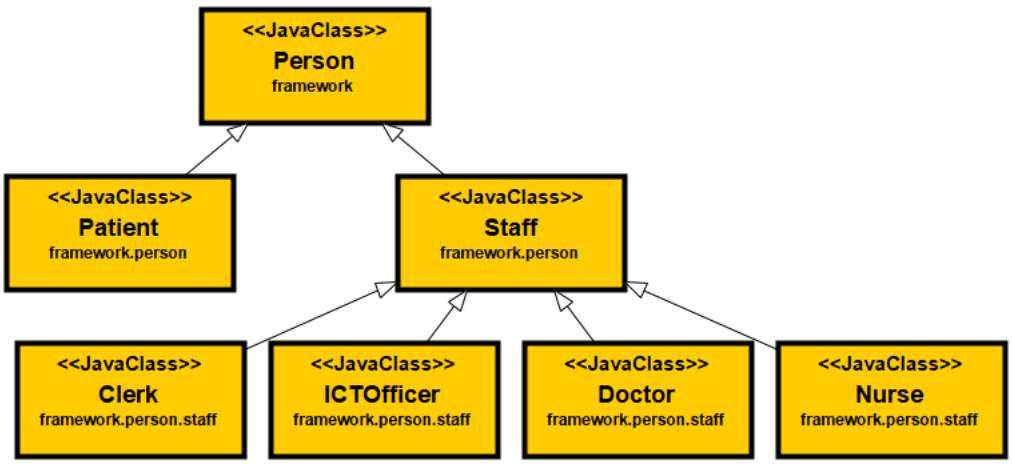


Figure 4.1: Simplified UML over the person structure

In the UML diagram shown here in figure 4.2 the hospital structure is displayed. It is almost the same as the Person structure besides that the class beds is associated with the class InPatientDepart. This is because the InPatientDepart should know that the specific bed stays in that department but the class bed does not need to know what department it is in. Further more this relation is also found between the hospital and the abstract class department.

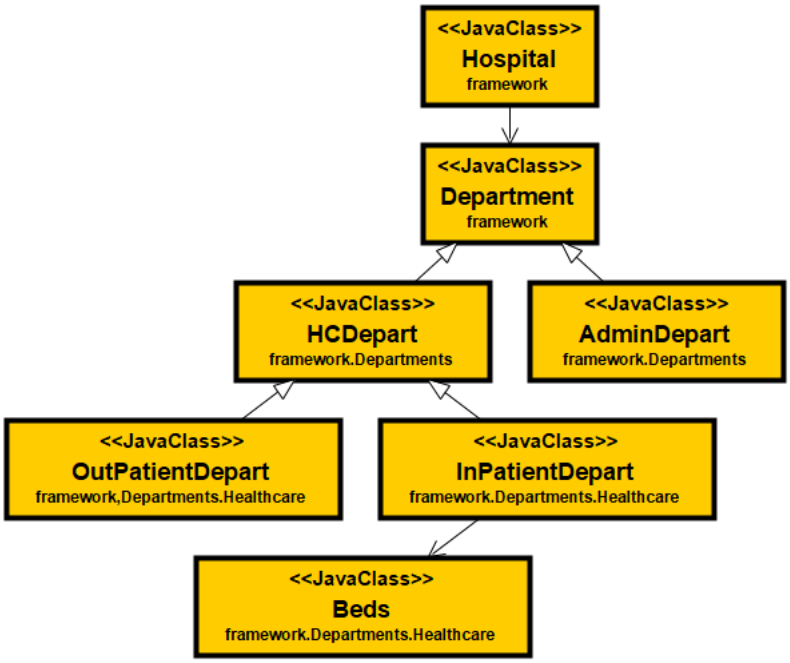


Figure 4.2: Simplified UML over the hospital structure

The last UML diagram in figure 4.3 shows the structure of all the function classes that is associated to the hospital where the API class is the large liaison that controls all actions done on the hospital.

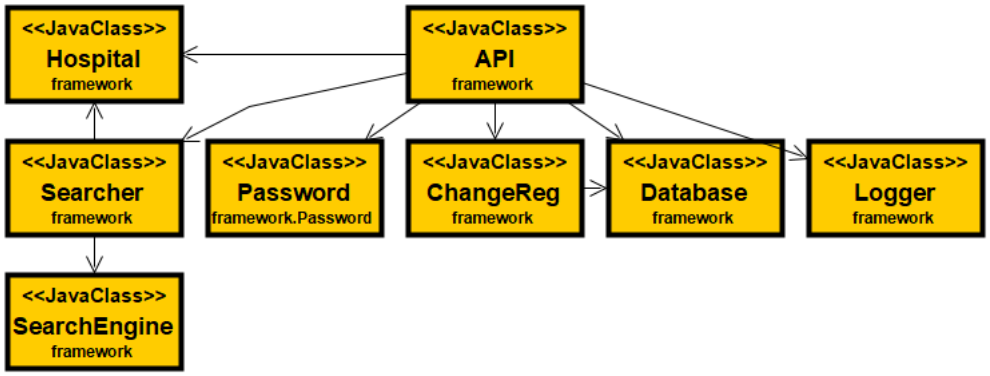


Figure 4.3: Simplified UML over the functional structure behind the hospital

4.2 Design Patterns

Design patterns are used to structure a general path of the development of a code. If no patterns are followed the different developers on a project might develop in 10 different directions and tons of merge conflicts and other problems emerges. Therefore we have used different design patterns in our project that will be described in detail in the following section.

Singleton API

There is one main design pattern in our program, namely the way we create our API. In order to insure the security of our data, we have decided to create one single class that handles all communication between the GUI and the database. We did this by creating an singleton API, which holds an instance of a hospital, containing

all our departments etc. Doing it this way, it was possible to avoid making any other objects singleton object, which would have made exposed to java dependency injections. By not making any singletons besides the API, only the API holds the pointers to the object generated by itself. In this way, data-flow in and out of these object can only happen through the API itself. This way of constructing our system made it possible to monitor all data-flow from database to user and vice versa. All method in the API also took a userID and password as inputs, which enabled us to check if a user actually was allowed to perform the requested operation. It is worth mentioning that the only output the API gives is in string format; no pointers. In this was, there is no way a user can circumpass the API and interact with an object directly.

Factory add staff

We have used the factory design pattern when creating new staff members. Instead of exposing the staff subclasses (ICTOfficer, Clerk, Nurse, and Doctor) we create them through a method, which takes a String input that is matched against the name of these classes. This ensures the strict ownership we have tried try implement, where the user should not be able to ever interact with an object directly, besides the API.

Observer listener

The observer listener pattern has been used in the way we are selecting what type of department we want to create one under the functionality "Add department". We have used radio buttons to ensure that one and only one department type is always selected. Additionally we have implemented, that the input field that takes the number of beds is only editable iff. the in patient department type is selected. This was implemented by letting the in patient department button send out a message saying that it was selected, and then enabling the field and the other way around when it was deselected.

Adapter

When connected to the SQL database, we used a driver package as an adapter, we did not create one our selves. However, before we could use the driver, we had to create our own little piece of software, that transformed our data into something the driver could upload to the database. In that way we had to implement a adapter between our main program and the driver we used to communicate with our database. Additionally one could argue that the API serves as an adapter from the GUI to the model, and enables a user to easily do allowed operations on the data.

Model View Controller

The GUI is set up by following the "Model, View, Control" design pattern. This design pattern makes sure that the interface doesn't see the model of the program but only the controller that disseminate the information got from the user through the view, to the back-end which is here called the model. In our program we have a class called Session and a class called User in the package model. These passes on who has logged in to the system and makes sure that the user only sees what he or she has permission to see. Functions further down in our model, is controlled by an API which is called in all the controller classes in the GUI with information from the user.

4.3 Graphical User Interface

The Graphical User Interface has been implemented by following the MVC pattern. The user logs in to the program and then a session is passed around the different pages in the program holding the user information so the user only is shown what he or she is allowed to see. This is done by calling the session through the controller of a specific view and this way only showing the specific buttons the user has permission to see. Further more our GUI also calls no functions in the view but the view calls the controller that then disseminate the information to the back-end (model). Here our program actually deviates a bit from the model view controller. Here our program is more like the MVA (model view adapter) that instead of having a triangular structure between model, view and controller has a linear structure with view, controller and then an adapter called an API. It then passes on the Strings got from the user to the back-end. This way we make sure that no objects in the back-end sees the GUI. In this way we have also added a double check of the users clearance to ensure only people allowed to use specific functionalities can use them. This is done by retrieving user information in the API to ensure if the user querying a functionality actually has permission to querying it.

In our set up of the view we took the decision that the user should be able to see several functional pages at once. Therefore the program closes the main menu when you navigate to one of the sub menus but it does not close the sub menu when you navigate further to one of the funtionalities. The reason for this decision is that when using the program it can be nice to have the search patient functionality open simultaneously with the edit patient for exampel.

4.4 Refactor

Refactoring is another word for optimization or simplification of a code. When programs grow the code can have a tendency to get more and more complicated and unmanageable. To prevent this from happening you do what is called refactoring. In the following section examples of some refactorings we made in our code are explained.

4.4.1 Front-end

GridBagLayout refactor

Figure 4.4 shows how we were setting up the layout of all the buttons, labels, text fields and so on in our GUI. As it can be seen from the figure we were copying the same commands over and over. Therefore we changed it to using a class called GridBagLayoutUtils with an overloaded method called constraint. This method sets the different components correctly up in the grid as it can be seen from figure 4.5.

```
////////////////////////////// 2. linje

gc.gridx = 0;
gc.gridy++;
gc.weightx = 1;
gc.weighty = 0.1;
gc.fill = GridBagConstraints.NONE;
gc.anchor = GridBagConstraints.LINE_END;
gc.insets = new Insets(0,0,0,5);

inputArea.add(surnameLabel, gc);

gc.gridx = 1;
gc.anchor = GridBagConstraints.LINE_START;
gc.insets = new Insets(0,0,0,0);

inputArea.add(surnameField, gc);
```

Figure 4.4: Here the gridbag layout is defined without use of a method as done everywhere before the refactor

```
////////////////////////////// 2. linje
y = y+1;

inputArea.add(surnameLabel, GridBagLayoutUtils.constraint(0, y, 1, 0.1, 0, 0, 0, 5,  GridBagConstraints.LINE_END));

inputArea.add(surnameField, GridBagLayoutUtils.constraint(1, y, 1, 0.1, 0, 0, 0, 0, GridBagConstraints.LINE_START));
```

Figure 4.5: Here the gridbag layout is defined by use of a central method

Job type refactor

Seen from figure 4.6 the user was enable to give what ever he or she wanted as job type to a new staff member. In our hospital the only job types available are Doctor, Nurse, Clerk and ICT Officer and therefore the user could get error messages this way. This was not a problem for the system because it just returned an error message but all these user errors could be avoided by limiting the job type inputs. From figure 4.7 it can be seen we limited the user input to only existing job types by introducing a JComboBox with only existing job types.

```
private JTextField jobField;

jobField = new JTextField(11);

String job = jobField.getText();
```

Figure 4.6: Here the code for a JTextField is shown

```
private JComboBox jobField;
private String[] jobtypes = {"","Nurse", "Clerk", "Doctor", "ICT Officer"};

jobField = new JComboBox(jobtypes);

String job = (String) jobField.getSelectedItem();
job = job.replace(" ", "");
```

Figure 4.7: Here the code for a JComboBox is shown

Assign staff member refactor

Before we made this refactor one needed to enter alot of information to assign a staff member to a department as it can be seen from figure 4.8. This is cumbersome for the user and totally unnecessarily when all staff members has a unique ID. Therefore we changed the functionality to only need a department name and an ID as it can be seen from figure 4.9.

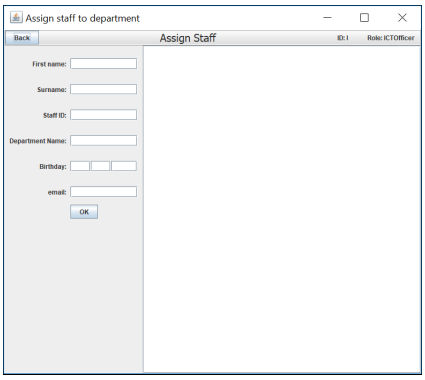


Figure 4.8: Here the GUI page for assigning a staff member before the refactor is shown

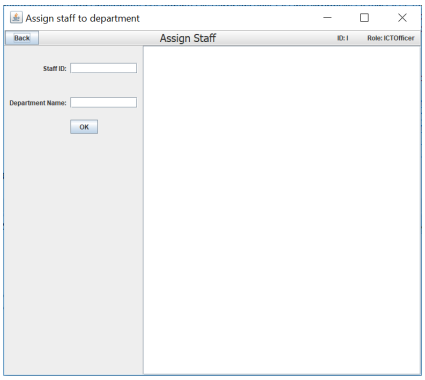


Figure 4.9: Here the GUI page for assigning a staff member after the refactor is shown

Enum refactor

Figure 4.10 and 4.11 shows the code without using enums. Here we are using strings to give different users, different clearing groups in the system. The weakness of this is that when writing the ID of a group as it is shown from figure 4.11 you get no information if you wrote the right ID. This is a huge problem when the code grows and the ID's should be used by several people. Therefore we changed the ID system from using strings to enums as it can be seen from figure 4.12. Here we are defining a group called jobtypes and when you want to get an ID from the group jobtypes, you just write jobtypes.xxxx and you will always know if you got the right ID.

```
public void setRole(String Role) {
    char first = Role.charAt(0);

    if (first == 'D') {
        role = "Doctor";
                }
    if (first == 'C') {
        role = "Clerk";
    }

    if (first == 'I') {
        role = "ICT-Officer";
    }

    if (first == 'N') {
        role = "Nurse";
    }


}
```

Figure 4.10: Here it is shown how the clearance of the users was checked before the refactor

```
if(session.getRole() == "Clerk" || session.getRole() == "ICT-Officer" ) {
buttonsPanel.add(btnRegister, GridBagLayoutUtils.constraint(2, 1, 0, 0, 0,10,50,10));
buttonsPanel.add(btnAssign, GridBagLayoutUtils.constraint(4, 1, 0, 0, 0,10,50,10));
```

Figure 4.11: This is a example of how the checks was made before the refactor

```
public enum JobTypes{
    Doctor,Clerk, ICTOfficer , Nurse;
}


public void setRole(String Role) {
    char first = Role.charAt(0);

    if (first == 'D') {
        role = JobTypes.Doctor;
                }
    if (first == 'C') {
        role = JobTypes.Clerk;
    }

    if (first == 'I') {
        role = JobTypes.ICTOfficer;
    }

    if (first == 'N') {
        role = JobTypes.Nurse;
    }
```

Figure 4.12: Here it is shown how the clearance of the users was checked after the refactor

4.4.2 Back-end

Singleton refactor

Before this refactor, both the database class and the password class were made as public singletons. This made no sense after we chose to pursue the API-singleton design pattern. The whole idea of having everything go through a single object, the API, which is visible for everyone is that all data-flow can be monitored and checked in one location. Therefore it is quite essential that it must be impossible to access any object used in the API without going through the API itself.

```
32    private API (){
33        // CONNECTION TO DATABASE TO ENSURE CONNECTION
34        DB = Database.getInstance(Database.DEFAULT);
35        Pas = Password.getInstance();
36        searcher = new Searcher(h);
37        R = new ChangeReg();
38        Pas.addPassToMap("I", "I");
39        log = Logger.getInstance();
40        log.write("SYSTEM","REBOOT","NONE");
41
```

Figure 4.13: Code-snippet from the API constructor before this refactor.

```
32    private API (){
33        // CONNECTION TO DATABASE TO ENSURE CONNECTION
34 //     DB = new Database(Database.DEFAULT,"jdbc:mysql://localhost:3306/","mydb","root","AGILE2019");
35        DB = new Database(Database.REMOTE,"jdbc:mysql://www.remotemysql.com:3306/","0S1l397yKA","0S1l397yKA","ceoLj1fgBZ");
36        Pas = new Password();
37        R = new ChangeReg(this.DB);
38        log = new Logger("ParticipationLists");
39        Pas.addPassToMap("I", "I");
40
41        log.write("SYSTEM","REBOOT","NONE");
42
```

Figure 4.14: Code-snippet from the API constructor after this refactor.

Conclusion

The creation of the program threw our team into a maelstrom of obstacles. We however adapted quickly to every new situation, and we realized that our work mindset needed tuning. After rethinking our work flow and implementing Agile principles our productivity skyrocketed. We touched upon many different production patterns development- and management wise, which greater improved the quality of our work flow. We tried to stay in the scope of the project requirements but at times made small miss steps. These miss steps in our work flow came at a cost - it resulted in moderate time being wasted. However, without these failures during our work, we would not end up with the product we have now. The product fulfills the purpose of the assignment and it follows the agile demands we set to please.

Appendices

Work Distribution

Merge Master
Karl Takeuchi-Storm

- Project Management
- User Stories
- Program Structure
- Department Management
- Tooling
- BDD-Procedure

Sprint Whip
Andreas Engly

- Database
- Staff Management
- Patient Management
- Tooling
- BDD-Procedure
- User Manual

Test Marshal
Mathias Søndergaard

- Graphical User Interface
- User Management
- Department Management
- Refactoring
- API
- Staff Management

Chief Architect
Anton Ruby Larsen

- Graphical User Interface
- Department Management
- Staff Management
- Patient Management
- Refactoring
- API

Refactoring Specialist
Mads Esben Hansen

- Refactoring
- Department Management
- Change Register
- Search Engine
- Change Log
- Participation List

Figure A.1: List of main tasks

User Manual

User Manual - Mojn X2000

B.1 Patient Handling

B.1.1 Registration of Patients

Upon first time arrival at the hospital, a patient needs to be registered in the system before anything else can be done. To do this, you must:

1. Click on "Patient"
2. Click on "Register Patient"
3. Enter personal information of the new patient
4. Click "OK"

Now you will receive a massage saying that the new patient has been registered. The patient has now been added to the system, and you should be able to find them in the system.

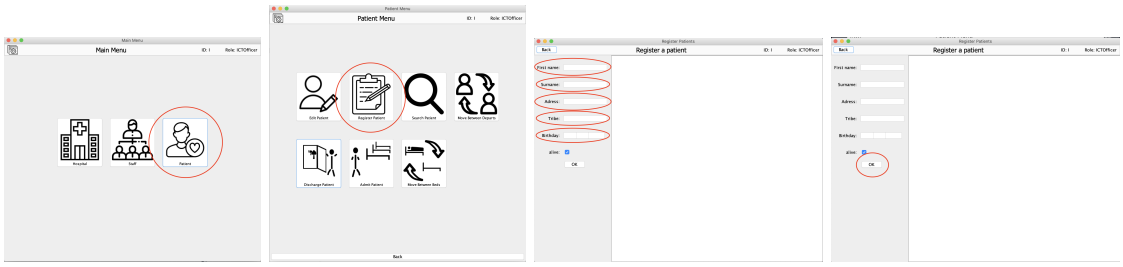


Figure B.1: Follow these simple steps to register a new patient

B.1.2 Admission of Patients

When a patient in the register has to receive treatment, he or she needs to be admitted to a specific department.

1. Click on "Patient"
2. Click on "Admit Patient"
3. Enter ID of the patient and the name of the department you wish to admit the patient to. If the department is an out-patient department, remember to specify the the triage level of patient. In this way the patients with the highest triage level will be treated first. If not specified, the patient will get lowest priority.
4. Click "OK"

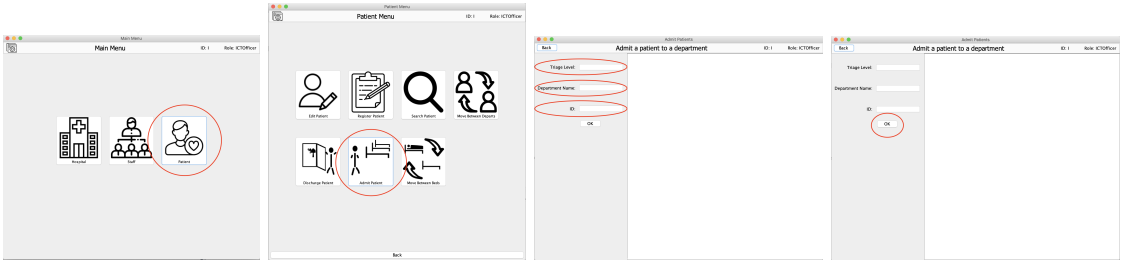


Figure B.2: Follow these simple steps to admit a patient

B.1.3 Searching for Patients

1. Click on "Patient"
2. Click on "Search Patient"
3. Provide information in the text-fields. If no information is entered, the search will match all patients.
4. Click "OK"

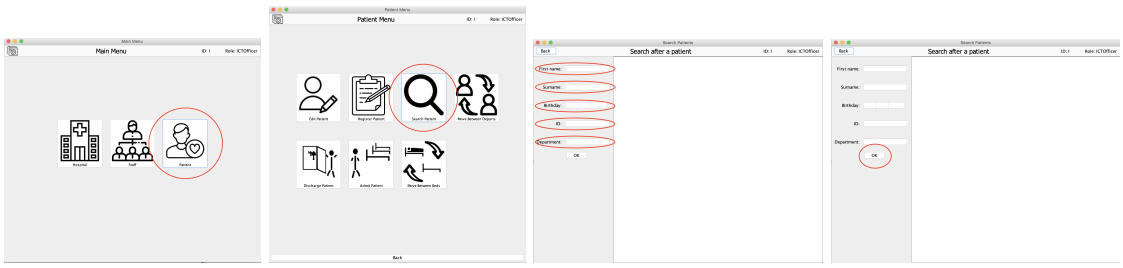


Figure B.3: Follow these simple steps search for a patient

B.1.4 Discharge patient

When a patient is admitted to a in- or out patient department, you might want to discharge him. This can be done following these easy 4 steps:

1. Click on "Patient"
2. Click on "Discharge patient"
3. Enter the patient ID of the patient
4. Click "OK"

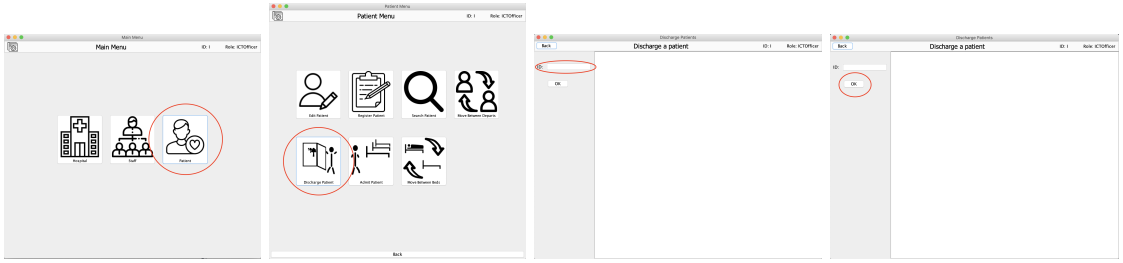


Figure B.4: Follow these simple steps to discharge a patient

B.1.5 Moving of Patients between Departments

This feature should be used whenever a patient, who is already admitted to a department, needs to be moved to another department to receive a different kind of treatment.

1. Click on "Patient"
2. Click on "Move Between Departments"
3. In the text-field "ID" type in the ID of the patient you wish to admit. In the text-field "New department name" you should type in the name of the department you want to move the patient to. If the department is an Out-Patient Department, specify the triage level. The higher triage level, the faster the patient will be taken care of.
4. Click "OK"

B.1.6 Moving of Patients' bed

When a patient is admitted to a department, he will be assigned to a random bed. If the patient needs to be moved to another bed, follow these easy steps:

1. Click on "Patient"
2. Click on "Move Between bed"
3. Enter desired bed no and the patient ID
4. Click "OK"

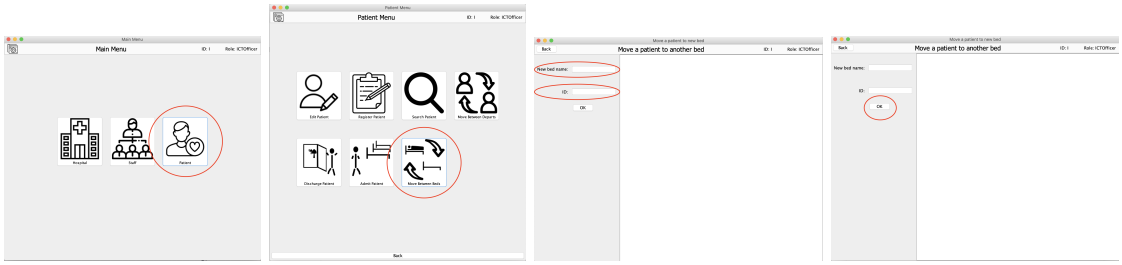


Figure B.5: Follow these simple steps to move a patient to another bed

B.1.7 Next Patient in Line

In a out patient department, you might want to get the next patient in line, this can be done by the system using these simple steps(obs, the patient is discharged automatically):

1. Click on "Patient"
2. Click on "Move Between bed"
3. Enter desired bed no and the patient ID
4. Click "OK"

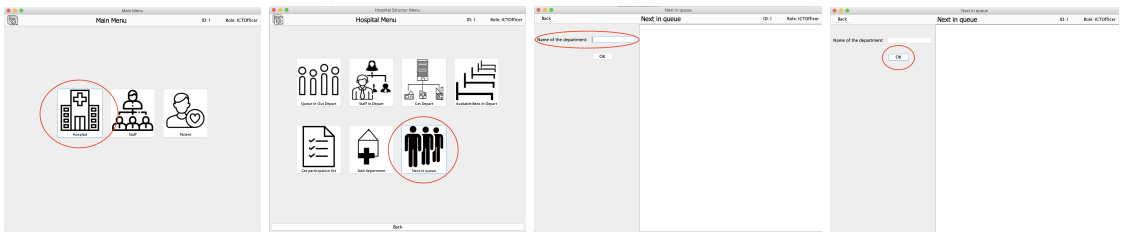


Figure B.6: Follow these simple steps to get the next in queue

B.1.8 Getting the Entire Queue

You might want to access the entire queue of a out patient department. Therefore there is a simply function that displays the queue directly in the program. To get the queue you need to follow these four steps:

1. Click on "Hospital".
2. Click on "Queue In Out Depart".
3. Enter the name of the department.
4. Click "OK".

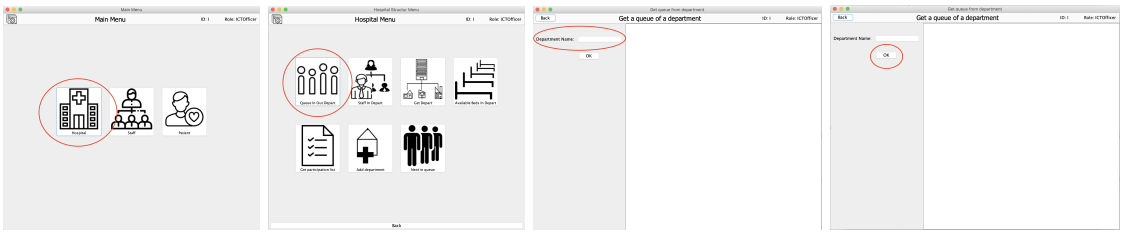


Figure B.7: Follow these simple steps to access the queue of an out patient department

B.1.9 Participation List

To get an customizable participation list that can directly be opened in e.g. Excel follow these steps:

1. Click on "Hospital"
2. Click on "Get participation list"
3. Enter properties of wanted participation list
4. Press "OK" The file will now to accessibly in the "ParticipationLists" folder in your home directory. If it is the first time getting a participation list, you must contact a system administrator, and get them to make sure there has been created a folder in your home directory with the name "ParticipationLists".

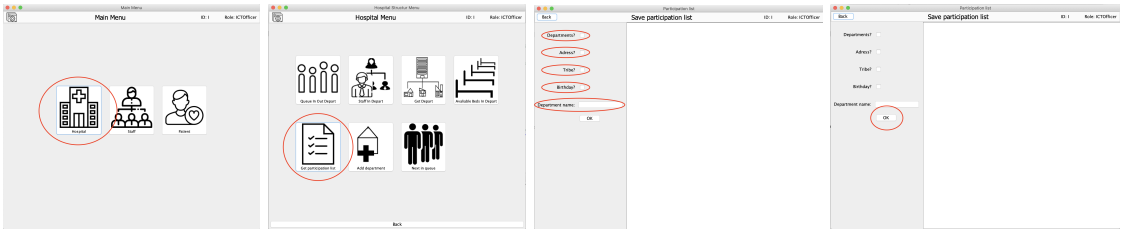


Figure B.8: Follow these simple steps to access the queue of an out patient department

B.2 Staff Handling

B.2.1 Registration of Staff

When a new staff member is employed, they must to registered in the system, to do this, follow these steps:

1. Click on "Staff"
2. Click on "Register Staff"
3. Enter the corresponding information in all of the text-fields.
4. Click "OK"

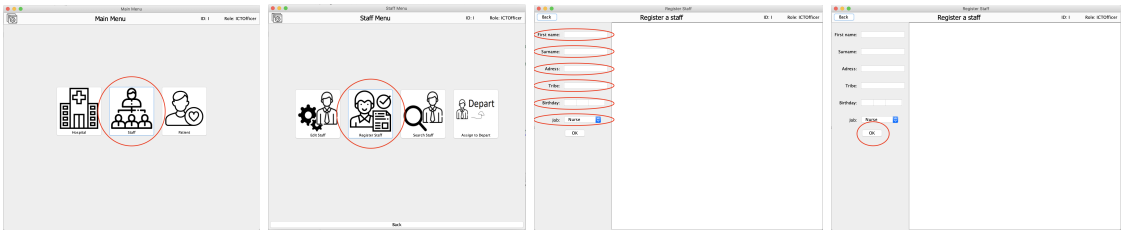


Figure B.9: Follow these simple steps to register a new staff member

B.2.2 Searching for Staff

You might want to conduct a search for a staff member to get their email address etc. To do this follow these steps:

1. Click on "Staff"
2. Click on "Search Staff"
3. Fill out the known information. In case all of the text-fields are empty, all staff members will be displayed.
4. Click "OK"

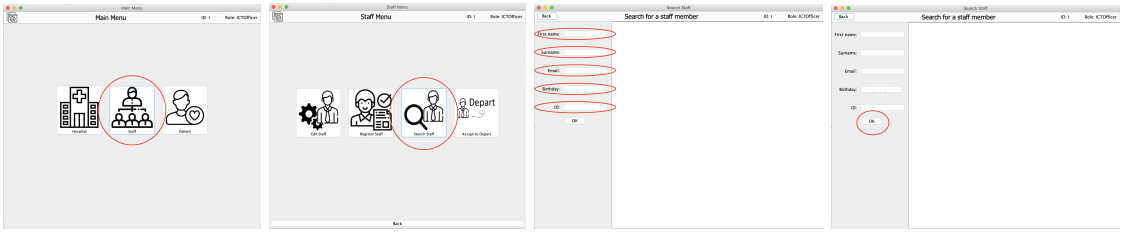


Figure B.10: Follow these simple steps to search for an existing staff member

B.2.3 Changing a staff

If you want to change the personal information or jobtype of a staff member, follow these four steps:

1. Click on "Staff"
2. Click on "Edit Staff"
3. Fill out the properties that shall be changed. Always fill in the staff ID (this cannot be changed).

4. Click "OK"

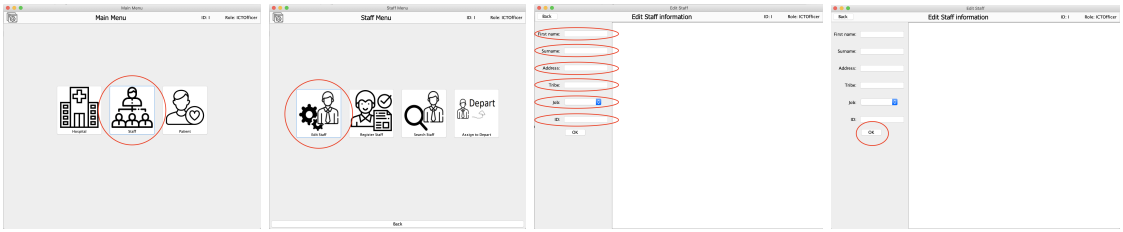


Figure B.11: Follow these simple steps to edit existing staff member from their ID

B.2.4 Assign a staff member to a department

When a staff member has been registered they need to be assigned to a department, to do this, follow these steps:

1. Click on "Staff"
2. Click on "Assign to depart"
3. Enter the staff ID and wanted department
4. Click "OK"

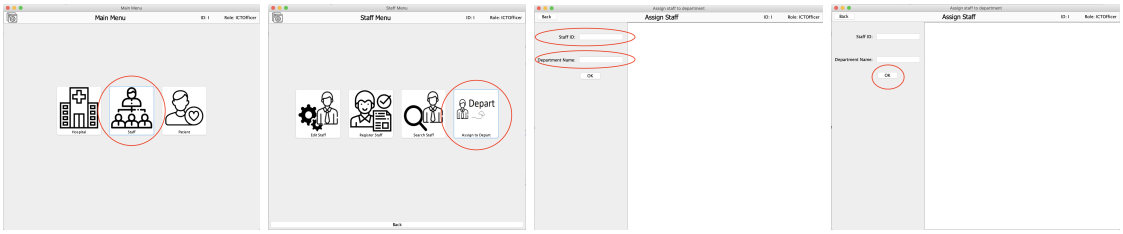


Figure B.12: Follow these simple steps to edit existing staff member from their ID

B.3 Department handling

B.3.1 Adding a New Department

In case the structure of the hospital needs to be changed, you might want to add a new department.

1. Click on "Hospital"
2. Click on "Add department"
3. Enter name and the type of the new department. In case the department should be of type in-patient, the maximum number of beds must be specified.
4. Click "OK"

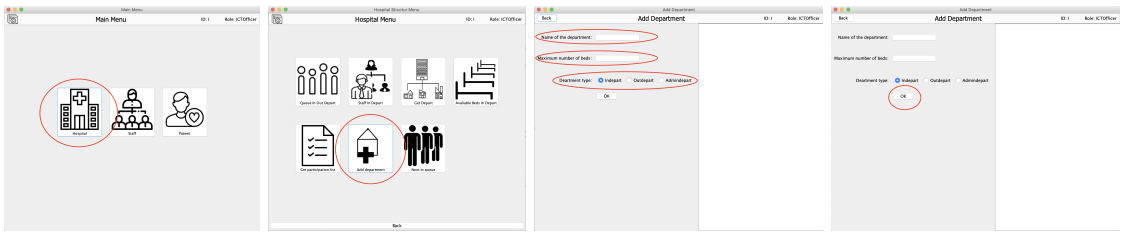


Figure B.13: Follow these simple steps to add a new department to the hospital

B.3.2 Searching for Staff in Specific Department

You might want to be able to get a list of all employees assigned to a specific department, to do this follow these steps:

1. Click on "Hospital"
2. Click on "Staff in Depart"
3. Enter the name of the department
4. Click "OK"

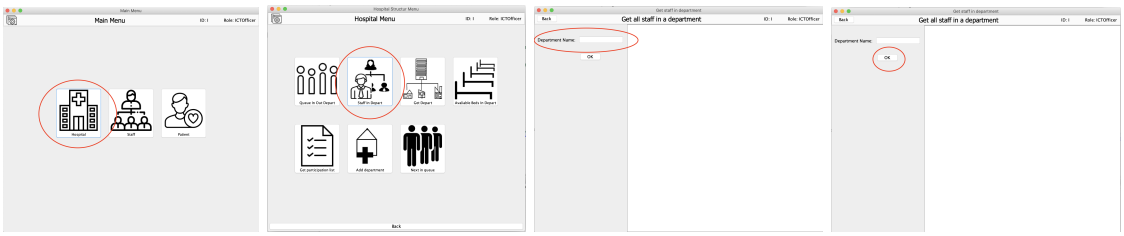


Figure B.14: Follow these simple steps to get a list of all employees working in a specific department

B.3.3 Get all departments

If one wants to find the names of all departments follow these three steps:

1. Click on "Hospital"
2. Click on "Get Depart"
3. Click "Get Depart"

Figure B.15: Follow these simple steps to get a list of all employees working in a specific department

B.3.4 Get available beds in department

If one wants to find available beds for an in department follow these easy steps:

1. Click on "Hospital"
2. Click on "Available Beds in Depart"
3. Enter the name of the department (only in-departments).
4. Click "OK"

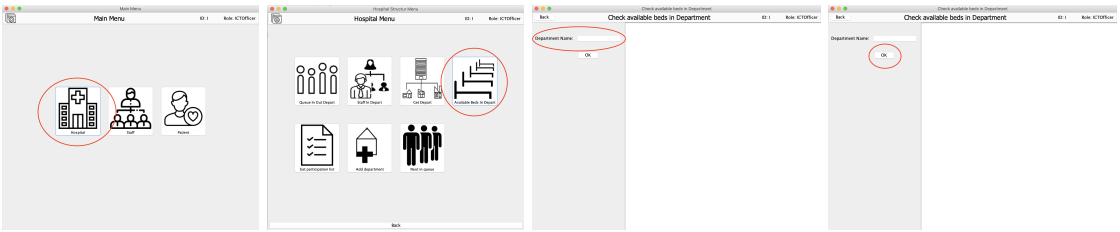


Figure B.16: Follow these simple steps to get a list of all employees working in a specific department

B.4 Login handling

B.4.1 Add login to new staff member

When a new staff memeber has been registered in the system, a login need to be added to that staff member. This is done following these four steps:

1. Click on the login menu (top left corner)
2. Click on "Add Password"
3. Enter the staff ID of the new staff member, and their wanted password
4. Click "OK"

After this procedure, it should be possible for the new staff member to login using their staff ID as user ID and their newly added password as password.

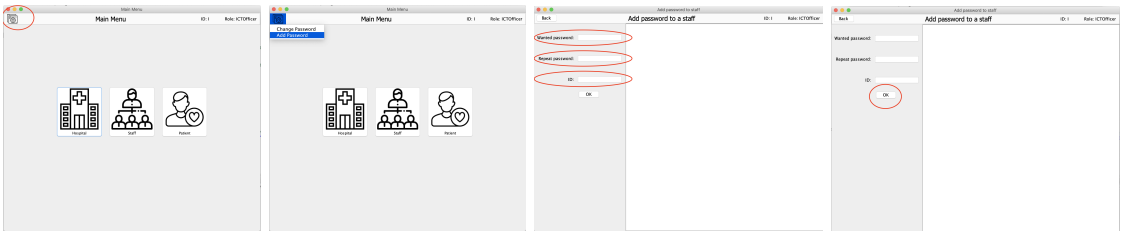


Figure B.17: Follow these simple steps to add login to a new staff member

B.4.2 Change login of existing staff member

A staff member might want to change their ID, to do this, follow these four steps:

1. Click on the login menu (top left corner)
2. Click on "Change Password"
3. Enter the staff ID, the wanted password, and the old password
4. Click "OK"

After this procedure, it should be possible for the staff member to login using their staff ID as user ID and their newly added password as password.

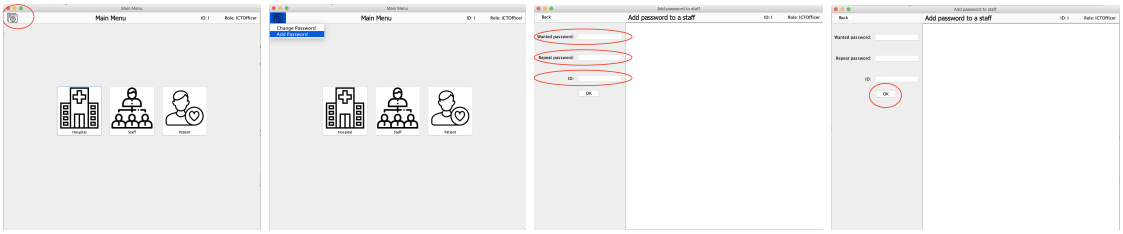


Figure B.18: Follow these simple steps to add login to a new staff member

B.5 Extra (Participation List setup and system log)

B.5.1 Participation list

When setting up the system on a new computer, a ICT officer needs to create a folder for the participation list. The program will try to locate the participation list in a folder called "ParticipationLists" in the computers home directory. Follow these steps to insure this feature will work correctly.
For windows:

1. Open cmd
2. Enter "cd C:\Users\%username%\ParticipationLists" If you get a message saying no such directory exits, enter the following: "mkdir C:\Users\%username%\ParticipationLists"

For mac:

1. Open terminal
2. Enter "cd"
3. Enter "cd ParticipationLists" if you get a message saying no such directory exits, enter the following: "mkdir ParticipationLists"

After this procedure, it should be possible to use the "Get participation list" feature in the program. All files created through this feature can be found by:
For windows:

1. Open cmd
2. Enter "explorer C:\Users\%username%\ParticipationLists"

For mac:

1. Open terminal
2. Enter "cd"
3. Enter "open ParticipationLists"

Now a folder should open containing the a .csv file with your participation list.

B.5.2 System log

When setting up the system on a new computer, a ICT officer needs to create a folder for the system log before you do anything else. This is done following these instructions:
For windows

1. Open cmd
2. Enter "cd C:\Users\%username%\ParticipationLists" if you get a message saying no such directory exits, enter the following: "mkdir C:\Users\%username%\ParticipationLists"

For mac:

1. Open terminal
2. Enter "cd"
3. Enter "cd ParticipationLists" if you get a message saying no such directory exits, enter the following: "mkdir ParticipationLists"

If needed the system log can be opened the following way:
For windows:

1. Open cmd
2. Enter "C:\Users\%username%\ParticipationLists\log.csv"

For mac:

1. Open terminal
2. Enter "cd"
3. Enter "open ParticipationLists/log.csv"

Project Plan

Danmarks Tekniske Universitet

02160 Agile Software Development:
Hospital Information Management Suite

26. marts 2019

OO Project plan - heisenbug

Authors:	***Study No:***
Andreas Heidelbach Engly	s170303
Anton Ruby Larsen	s174356
Karl Emil Takeuchi-Storm	s130377
Mads Esben Hansen	s174434
Mathias Jarl Jarby Søndergaard	s174426

C.1 Choice of Agile Framework

We will work with the SCRUM-framework, which will enable us to create a concrete schedule that can adapt if unforeseen issues arise. The timing on the sprints are based on a hourly rate, meaning we won't assign sprints to last for weeks, but "working hours" instead. In a SCRUM-framework the relationship between costumer and developer will often be central in the development of user stories and requirements. In our case, we will create the user stories ourselves from the way we have understood the requirements. The requirements will normally be specified in during negotiations between developer and customer. However, we do not have direct contact with the customer, and do not have the possibility of negotiations with them. The requirements will therefore be made through interpretation of the project and negotiations within the group.

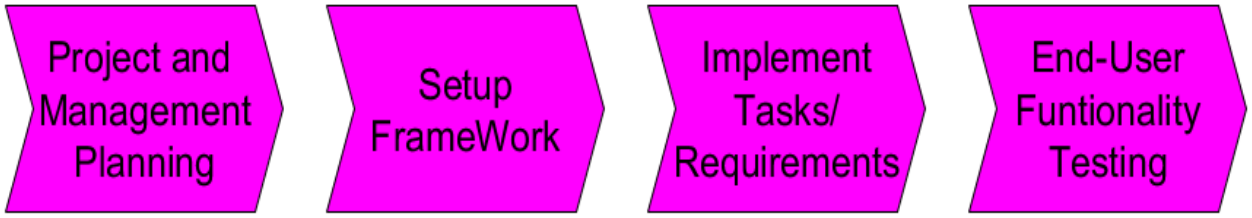


Figure C.1: Overview for Project Flow

C.2 Backlog Management

Our backlog will initially contain all mandatory- and optional assignments. The mandatory functionalities constituting the back-end of the system will have the highest priority. In our terms "business value" is determined as grades, thus mandatory assignments are considered highest business value, after this costumer value will also influence the business value, in the same manner as technical difficulty. Thus front-end development will take place once the basic mandatory functionality is implemented. The front-end will be an implementation of the GUI.

C.3 Task planning

All task planning is done at the "Retro Perspective" meeting. The main goal of these meetings is that everyone knows what to do and when it should be done. For instance going from requirements to scenarios is done through a group discussion, where everyone is allowed to make their points and tell their interpretation of the requirement. A "Retro Perspective" meeting will consist of four main elements.
1) Management process review.
2) Backlog re-prioritizing.
3) Re-factoring issues.
4) Sprint selection.

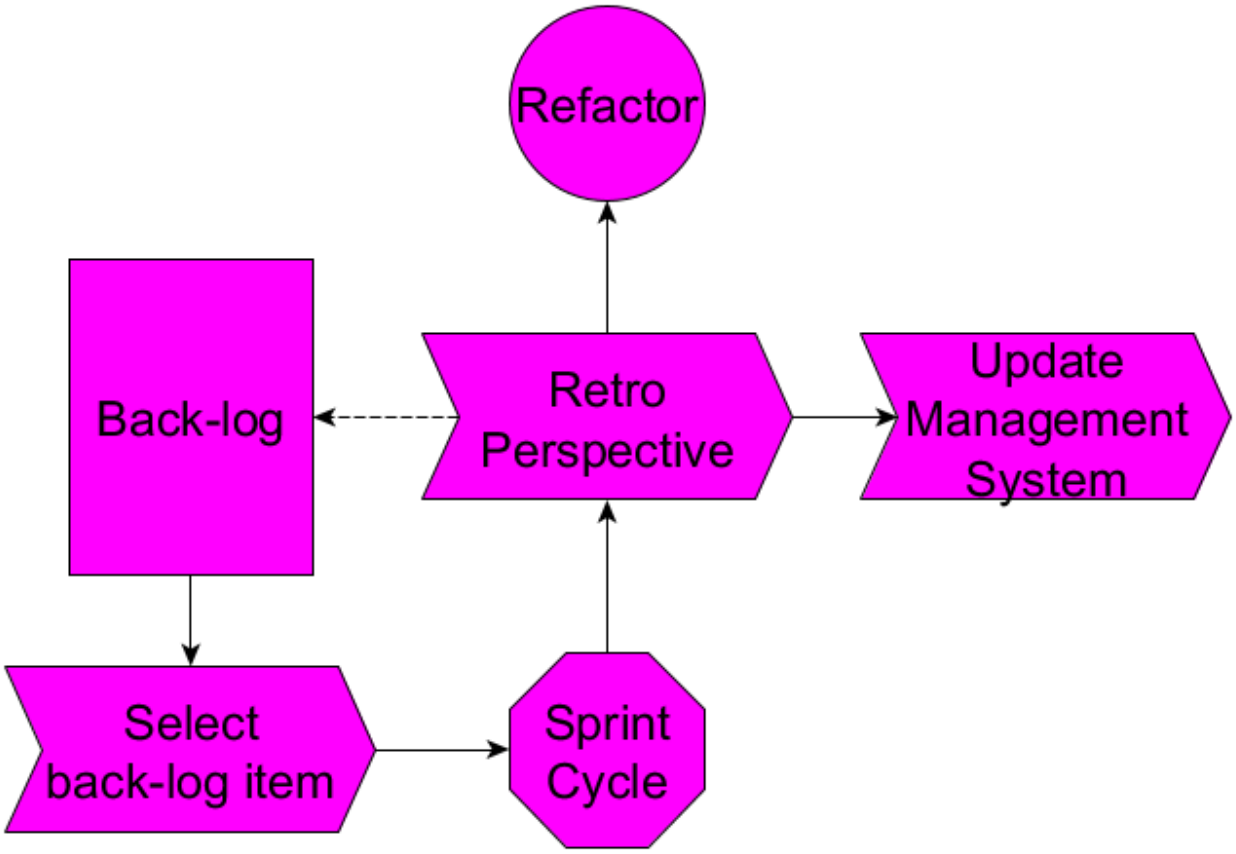


Figure C.2: Flowchart for Management Process

In addition, some of our task planning is done on the run. If some critical bugs appear, or very necessary refactors occur while doing a sprint, these shall be handled instantly and the team should be noticed about a critical change.

C.4 Definition of Done

We define a SCRUM-sprint to be done when we have finished the following things:

1) User stories.
2) Cucumber tests.
3) The module - implementation in the API.

However, not all sprints followed this definition of Done. We for example made one big GUI sprint, connecting the API to a front-end. This GUI sprint followed another definition of done:

1) View-class
2) Controller-class
3) Connection between API and controller class
4) Test if the functionality works.

C.5 Estimation and scheduling practices

To create a sustainable pedagogical schedule we will follow the SCRUM framework and assign a certain time frame for sprints. After every sprint we shall hold a meeting, discussing how things went. The key tool used is Trello, where a Kanban board is used for backlog management. It is clearly agreed that no more than two items can be in active sprint at any time. Further details are described in "Best Practice"-appendix. As the projected is limited in size, the Kanban chart is deemed sufficient for the purpose of burn-down. Estimating the exact time amount for sprints will be hard. Thus we have discussed to be flexible with the time frame from sprint to sprint. If a sprint requires more or less time than planned, adjustments to our guiding schedule will be made.

C.6 Test strategy

At the start of sprint on a module we define a BDD-test from the user stories. Our main goal is to fulfill these initial tests. When these tests are completed, we investigate every part of the module by asking and answering as a minimum the following questions:

1) Are there any obsolete parts?
2) Has the goal of the module changed? If yes, does our module fulfill this new goal?
3) Can some of the parts be rewritten?
4) Is it implemented in a secure way (access modifiers, stuck state etc.) ?

These questions will be answered with the help of JUnit- and Cucumber-tests.

In relation to GUI-testing, we can test the different functionalities by hand. We cannot apply a JUnit testing strategy to the GUI classes, as this makes no sense. Some discussion about creating a Click-bot has been made, but we deemed it overkill to implement such a tool.

C.7 Team agreement

We have agreed on doing our best and to be time efficient. Good communication is crucial, especially if someone is delayed or can't attend a meeting. In order to be efficient we strive to create a friendly and helpful working environment where we can utilize each other's abilities. Being flexible is also a feature we should all adopt in order to overcome sudden changes and challenges that may occur doing a sprint. Finally, we should be respectful. If one finds themselves discriminated against, it will reduce their productivity by a great factor. At last it is expected that every member of the group follows the "Best Practice" policy.

C.8 What is an Agile Mindset?

A loop based work process. We plan what to do a defined time period and then work on those specified tasked in smaller teams in the defined time. When the time is up we meet and show what we have done and what to do now.

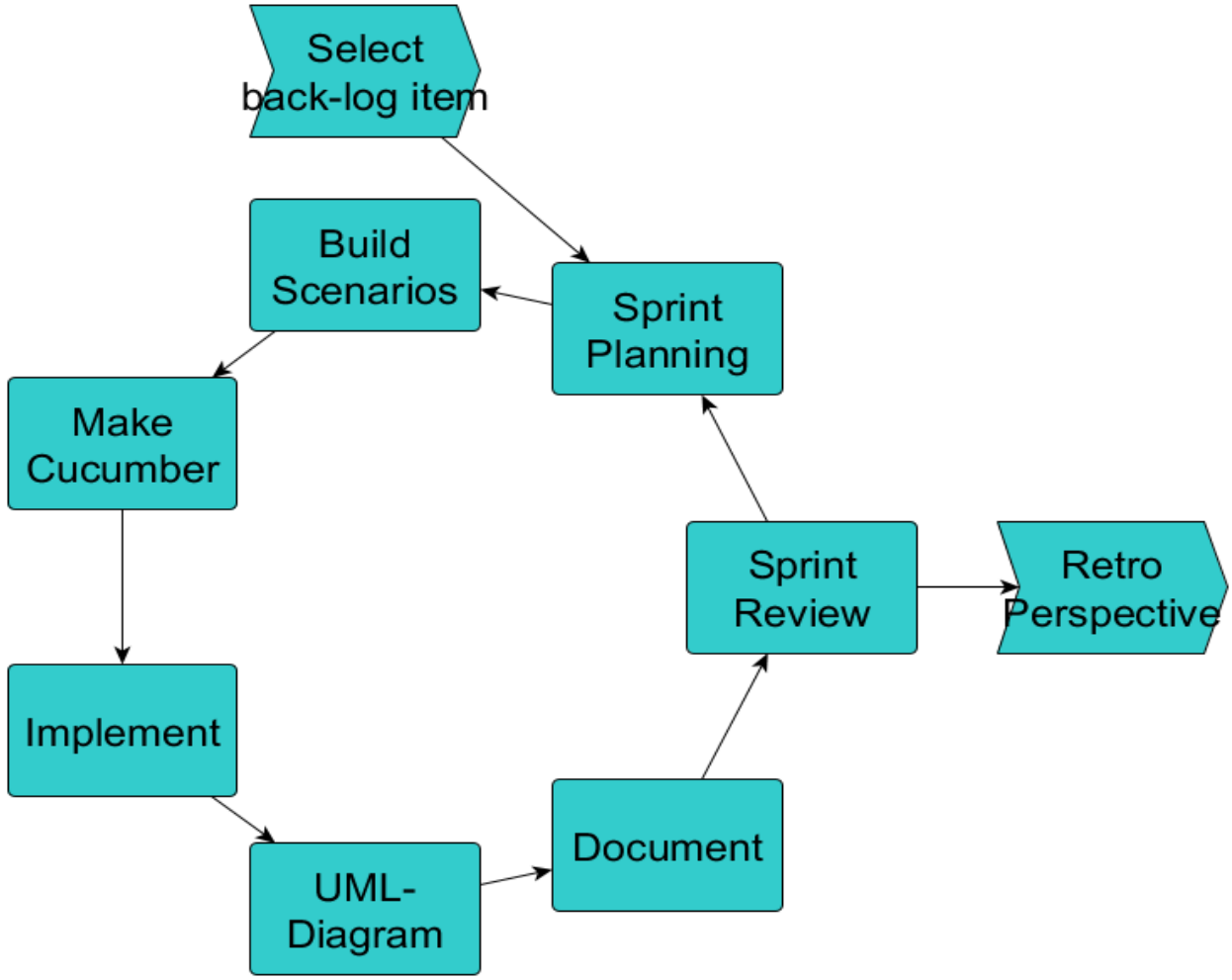


Figure C.3: Flowchart of our adapted sprint rotation.

This structure is repeated and always evaluated up against an overall plan. The management process is updated after each retro perspective meeting. If it wasn't in an educational setting the agile mindset would also include an ongoing customer contact. An example is user-stories. These would have been developed with the costumers to prevent that the final product does something completely different from what the costumer demands. We try to implement this by discussion internally in the group.

Each sprint contains six steps. Sprint planning, where the chosen sprint item, is divided into smaller tasks. Then the user story is made into a scenario, which again is used to produce a cucumber test.

Then enough implementation is done to pass the cucumber test, the overall UML-diagram is updated. Finally the commenting of the code is checked and updated to a high level. However, it should be mentioned that UML-diagrams are being low-prioritized. They should only be created if the implementation is abstract or difficult to understand.

C.9 Metrics and Continuous Improvement Practices

As described earlier we will use a loop structure in our work process, where we continuously discuss how things are going. By doing this we are able to move people around to where we need more people to accomplish certain tasks. Furthermore, we are able to adjust the plan if we find unexpected challenges. In this way, no groups will get ahead of the rest. If this happens they can work without knowing if the others run into trouble.

This can result in wasted work. Further both our management, agile- and internal structures must be updated continuesly.

Best Practice

Danmarks Tekniske Universitet

02160 Agile Software Development:
Hospital Information Management Suite

27. April 2019

Best Practice - heisenbug

Authors:	***Study No:***
Andreas Heidelbach Engly	s170303
Anton Ruby Larsen	s174356
Karl Emil Takeuchi-Storm	s130377
Mads Esben Hansen	s174434
Mathias Jarl Jarby Søndergaard	s174426

D.1 BDD Work-Flow

D.1.1 User stories

A user story is a communication tool for the software developer to agree with costumers on features to implement. A set of user stories, describing all wanted features should be made in collaboration with the costumer, where every user story is unique, and independent of each other. In some cases several user stories can describe the same feature in case the feature is used in different ways.

Generating a specific user story

The format of a user story is very concrete, however the interpretation of the implementation is not. Once the user story has to be translated from a user story to a scenario, a firm structure is desirable, thus the following interpretation is adopted.

A user story is short consisting only of three parts. The first is a specific user to which it applies, if it applies to every user a more general term is used. This is used for the costumer to identify the users of the system. See below figure.

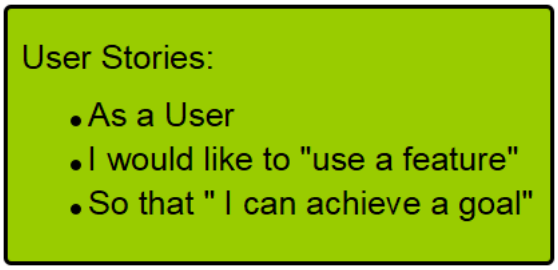


Figure D.1: User story implementation

The second part is applying a specific feature, this description is twofold, as it must be clear what the functionality of the feature is, to both the customer and the coder.

The last part describes the value of the feature to the customer. This should be a clear purpose of implementing it. The goal achieved by the user from above.

User Story to Back-log

Once the set of user stories is in place, only a little more work is needed. We need to specify a set of acceptance states for each user story. Here we have decided to implement three categories, each can contain several acceptance states, but should fall in one of the three.

A positive out come, in case of a search, this could be the one item that matches the search or a list of item matching said search.

A negative outcome, either reporting that the user did not use the system correct or that the given usage rendered something empty. ie. returns an empty search result.

The last type of outcome is an error report, where the system produces an software error.

Once all user stories has clear acceptance states, they are placed in the backlog, without any priority what so ever. Once the team reaches the point, of a backlog item selection meeting, the backlog item are chosen based on the business value, at the given time. The business value of an item can change, as depending on costumer needs, negotiated price etc.

D.1.2 Scenarios

Gherkin language

The basis for this next section is the gherkin language, please see this link for a clear definition. `https://docs.cucumber.io/gherkin/reference/`

Converting a User Story to a Scenario

This might seem like an trivial task, but a clear structure is given in the Gherkin language, and it is out most important to adhere to.

The goal is to generate an .feature file, which clearly describes the functionality of a feature. First a clear background is Given, When a user performs an action and maybe some more actions, Then a certain outcome is provided.

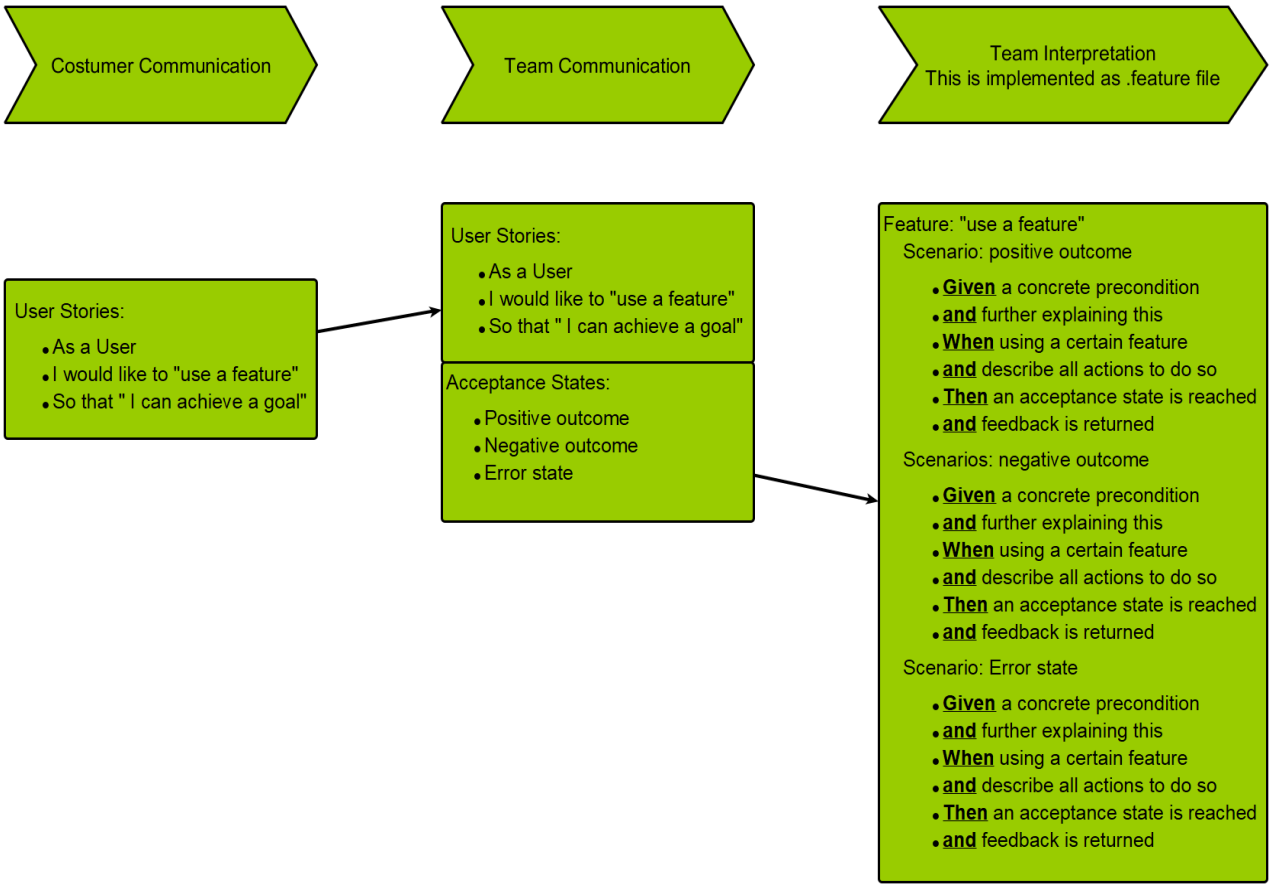


Figure D.2: From user stories to scenarios

D.1.3 From Scenarios to Implementations

Cucumber is a tool that supports Behavioral-Driven Development (abb. BBD). It means we develop with respect to certain behaviors of the program requested by the costumer. These behaviors are defined from the user stories.

All keywords (i.e. given, when, then) of the Gherkin source file (extension .feature) will be associated with an annotation test in a java-file in the stepDefinition-package. By convention the .feature-files should be named the same as the .java-files such that it is easy to see that they are related.

Avoid making the associated .java-file from scratch by creating a class FEATURE_NAME.java and go to the package "testRunners". Right click anywhere in the package and run as JUnit-test. The output will show which method implementations are missing. Copy the code snippets and insert it into the FEATURE_NAME class. After implementing the tests with the associated classes right click on the Maven-project and run as "Maven build...". In the goal field type "clean compile assembly:single". If it compiles, no conflicts have been introduced. If the compilation is successful, run as "Maven test". In that way it can be seen whether the tests are passed.

D.1.4 Commenting code

In an ideal world the code should be commented before being merged into the master. However, is the code is already readable, and easy to understand this is not an important criteria. The code should however at some point be commented.

D.1.5 Additional Material

Click the text

1. SCRUM: Lightweight process framework for agile development
2. How to create Cucumber Step Definition class
3. Gherkin: Language guide to cucumber

D.2 Applied Tools

D.2.1 Management Tools: Trello

Two boards are used in Trello, an project To-Do list and a Kanban board for the SCRUM process.

The items in the To-Do list should be kept down, however development is also an key element of this project.

Whenever a group/person works on a new backlog item, the item should be moved to *progress* by dragging it from *backlog*. While working on the different parts the item is moved through the Kanban chart, to keep the progress updated.

The scheduled group meeting will be visible under *calender* in the upper right corner.

The tool can be accessed here *trello*.

D.2.2 Version-Control System: Git using github.com

In order to keep development moving and reduce conflicts, each group member has been assigned a working branch. The work structure is simple, before writing any code re-base from master. then implement code and check test coverage. Then re-base, solve conflicts and merge.

When ever taking a break or stopping for the day, make sure to commit and push your own branch, this will help minimize development loss due to crashes of computers, corruption and developer illness.

Whenever working on new aspect of the backlog a the personal branch should be re-based.

Merge into master every time a new feature is ready and tests are working.

Should any problems arise regarding git, further clarification will be given by the appointed git master as needed.

D.2.3 Communication: Telegram

The communication between group members will be maintained through Telegram. It will be used for questions, scheduling, and for personal matters of which the group members must be notified. Specifications for this practice will be reviewed at each group meeting.

D.2.4 Ongoing Report Writing: Overleaf

Under *Mandatory* and *Optional* on Overleaf folders for each functionality can be found.

For each functionality added by a subgroup a *test-procedure* must be described. The test description must contain a test-coverage in percentage, how it has been tested, and lastly the subgroup's thoughts on why the tests provided are necessary.

A UML-diagram must be created for each functionality. In that way it will be much easier to combine them into the final UML-diagram. It is recommended to use *Signavio*.

When a sub project is finished, a user manual must be provided. It must be clear for any users without previous knowledge of using IT-systems how to use the program to full extend.

D.2.5 Maven: Build Automation

Maven is a comprehensive tool that can be used to build projects. A key feature is the ability to keep track of dependencies, which is crucial with respect to distribution of the project. Further the Maven tool can be used to test the software at compile time, thus creating an automated software development environment.

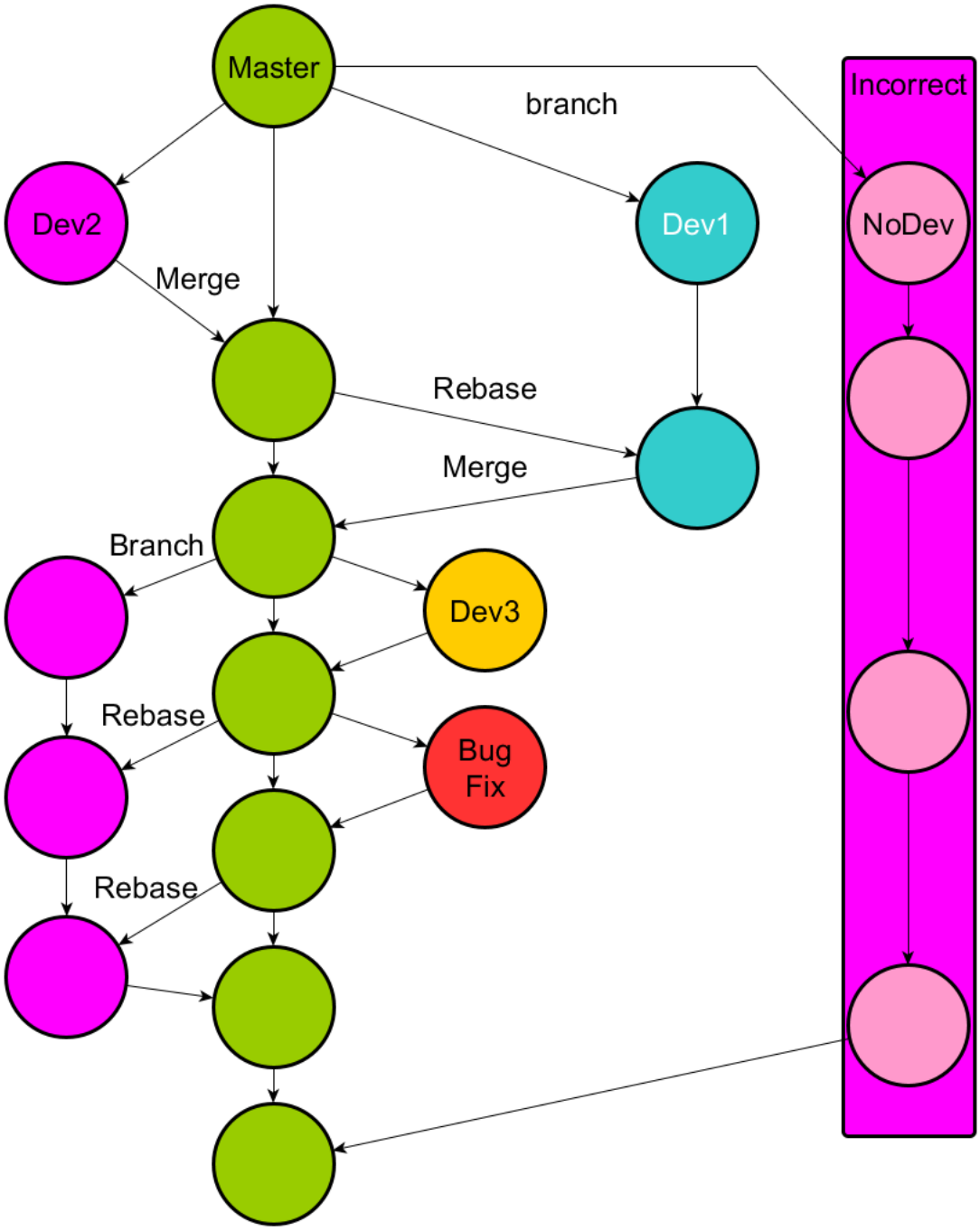


Figure D.3: Work-flow using Git(hub)

UML Diagrams

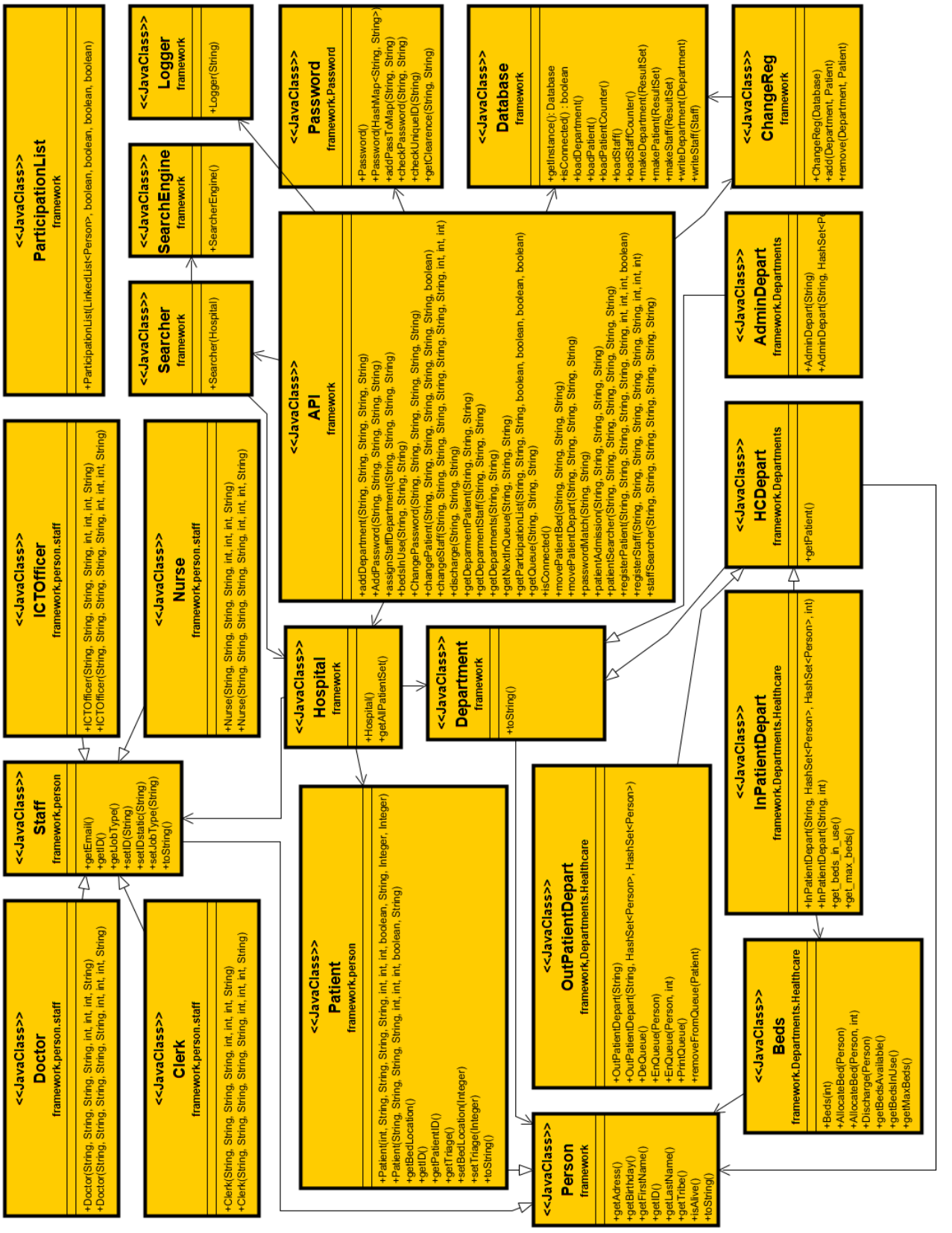


Figure E.1: UML of the entire framework

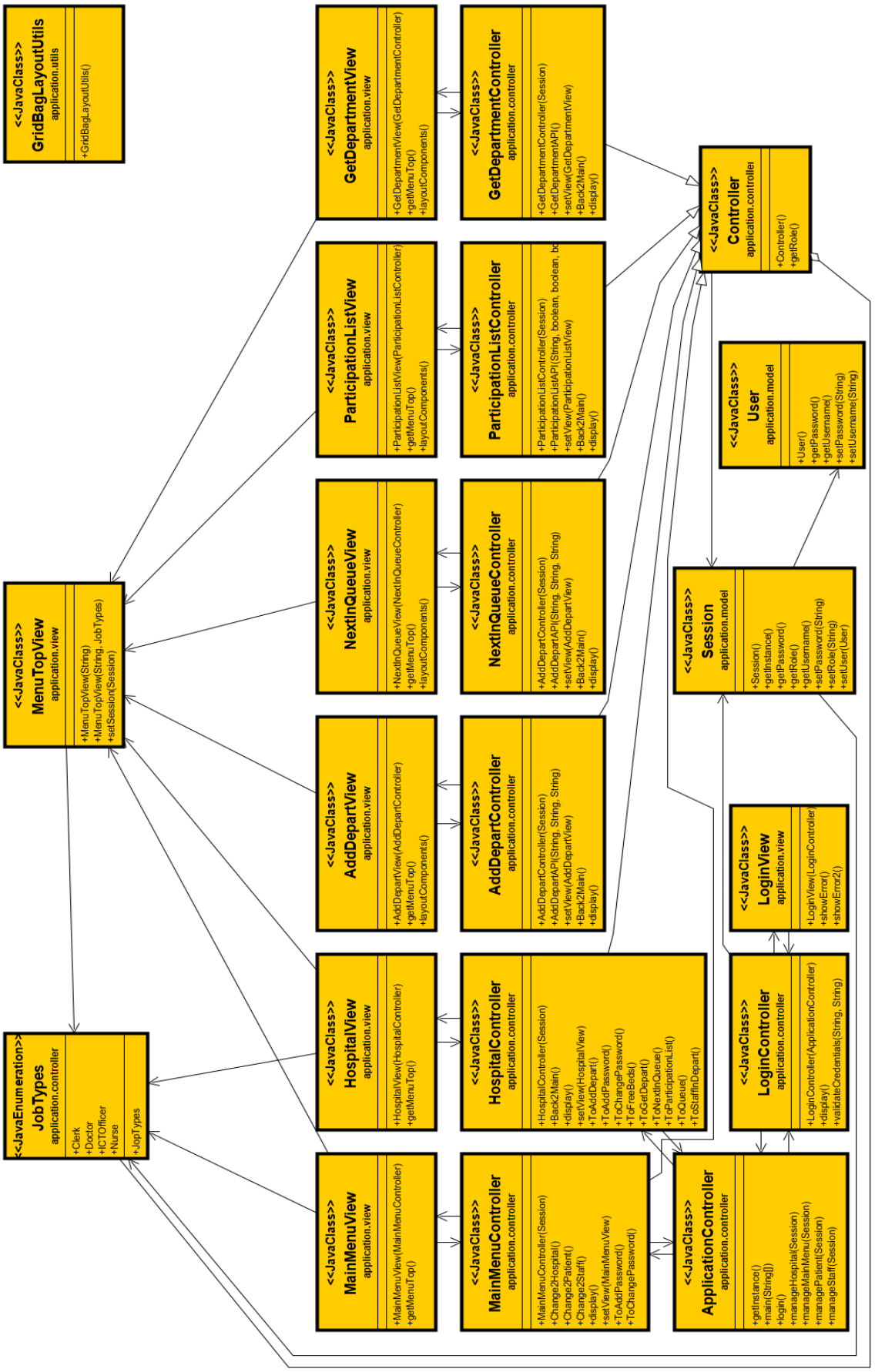


Figure E.2: UML of the Graphical User Interface (GUI)

User Stories

Danmarks Tekniske Universitet

02160 Agile Software Development:
Hospital Information Management Suite

26. marts 2019

OO User Stories - heisenbug

Authors:	***Study No:***
Andreas Heidelbach Engly	s170303
Anton Ruby Larsen	s174356
Karl Emil Takeuchi-Storm	s130377
Mads Esben Hansen	s174434
Mathias Jarl Jarby Søndergaard	s174426

F.1 User Stories - Features

Patients Records

As a Clerk I want to Register Patients so that they can proceed to get treatment.

- Patient registered.
- Insufficient data entered.

As an employee I want to Change Patient records so that we keep an updated system.

- Patient changes registered.
- Incorrect data entered.
- patient does not exists

As an employee I want to Search for Patients so that I can ensure they get correct treatment.

- Patient matching search.
- List of patients matching search.
- No patient matching search.

Staff Records

As an ICT officer I want to Register staff so that we can keep track of who is working here.

- clerk registered.
- Nurse registered.
- ICTOfficer registered.
- clerk registered.
- Insufficient data entered.

As an ICT officer I want to change staff records so that our system is always up to date.

- Employee does not exist.
- Staff changes registered.
- Incorrect data entered.

As an employee I want to search for staff members so that we can manage the staff.

- Staff matching search.

- List of staff matching search.
- No staff matching search.

Department Management

As a health care staff I want to check if beds are currently available, so that we can determine if new patient can be admitted.

- beds available
- No beds available

As a health care staff I want to retrieve the numbers of beds currently in use, so that we can manage how many patient in the department's beds.

- X beds currently in use.
- This department does not have any beds.

As an employee I want to admit a patient to a department. So they can proceed to said department for further diagnostics.

- patient admitted to in-patient department.
- patient admitted to in-patient department in bed no X.
- Bed no X not available.
- patient admitted to out-patient department.
- patient admitted to out-patient department with triage level X.

As a nurse I want to allocate a Patient to a bed so that they can receive treatment.

- Patient added to bed no X.
- Bed no X not available.

As a nurse I want to be able to move patient between beds for them to have an optimal recovery.

- Patient moved from bed no. Y to bed no. X.
- Bed no X not available.

As an employee I want to move Patients between departments So that they can get further treatment.

- Patient moved from department name Y to department name X.

- Bed no *X* not available at destination department.

As an employee I want to discharge patient so that we have room for new ones.

- Patient discharged.

As an out/patient doctor I want to be able to call the next patient in queue So that the patient receives treatment based on their turn and necessity.

- Patient *ID* called.
- No patients waiting.

As an employee of the out patient department. I want to see which patients are in queue so that we can allocate staff accordingly.

- List of queued Patients waiting.
- No patients waiting.

User Access Management

As an ICT officer I want to change any user's password. so that the users cannot see information they are not allowed to see.

- Password changed for staff *ID*.
- Access denied. Please check your user rights.

Persistency Layer

As a user of the Hospital Management System I want to restore the former state of the system upon restart.

- Successful boot of the system.
- Data is located in the correctly.

Participation List

As a clerk I want to be able to print a list containing *name* and *birthday* of patient in the department. So that the patients in the department can be verified.

- List in .csv format.