Testing Strategy

Dennis Thinh Tan Nguyen, Jacob Mullit Møiniche,
Thor Valentin Aarkjr Olesen, William Diedrichsen Marstrand

16. december 2015

Indhold

1 Introduction

This document will describe our reflection of the testing phase, our current implementation of our tests and the test we plan to incorporate in the future. We hope, by including tests in our early project stages will help us to better our design and catch errors which otherwise would have slipped past us. We firmly believes bugs and errors are easier to correct if the are caught early, and it will prevent them from snowballing and pick up size and scale.
We do not believe we are able to catch and eliminate all bugs and errors, but we have faith in a strong presence of tests will strengthen our products robustness

2 Unit Testing

We are mainly using unit testing to make it easier to detect and corrects faults, and to allow parallelism in the testing activities which is essential in our work, as every group member works on individual parts of the program simultaneously. We will incorporate Test Driven development, due to it helps us focus on testing, even from the earlier phases.

Black Box By testing all classes as they are developed, we make sure that all units are tested. The unit tests are automated which makes it easy to do regression testing, and in that way always have an idea of the robustness and quality of the program. We use black box testing to conduct tests of units before they are actually implemented. The reason for this is, that we know the functionality which the unit should support and as such we can create the tests based on this knowledge and check that our code does what is expected of it, by running the tests after the implementation. More specifically we use **equivalence testing** from black box testing to reduce the number of test cases. This has a huge impact in the cases where inputs and outputs are tested. In the equivalence testing we have give boundary testing high priority since it is often edge cases which make systems fail.

White Box After a feature is fully implemented it is then marked as ready for white box testing. The white box testing should be conducted by a team member who did not work on the implementation. This approach has been chosen to let other team members get knowledge of the code which they themselves did not work on. Furthermore, a tester who have no prior expectation to how the code should work might find errors the original writer would have fought of.

The goal for the white box testing is not to get path coverage, since this would be to comprehensive and not practically possible because of the many possible paths. Instead the goal is set to 90-95% branch coverage on essential components. The reason why this percentage is chosen and not 100% is that some cases will always be hard to cover, and thus this would require to many resources compared to the payoff from the tests.
The main goal is to test all major parts and all essential functionality of the program and a percentage of 90-95% would very likely achieve this.

Mocking To make sure that we can unit test classes that make use of the database without affecting the database we use mocking. By mocking the database we can simulate some of the database's functionality, and run the methods depending on it without ever calling the database. This means that the tests do not have any impact on the database, and as such we make sure that the database does not break when tests are run.

3 Integration testing

Unit Integration testing is a as the name apply a tests which components are working correctly together. It assume the the individual components are working correctly and is strictly testing for compatibility errors. This test approach allows us to test interconnectivity of classes instead of viewing them as isolated systems, and as a result gives us a broader image of which systems are effected of defect sub components.

3.1 Different types of integration testing

How to use integration test depends on your test methodology (Top down or Bottoms test strategy).

- **Top down test strategy** : implement integration testing by first testing the high level components integration and from there working downwards to the lower level components. This allows high level component and subsystems to be tested early in the development but comes with cost of delaying some test of the more complicated integrations test due to the lower level components have not been implemented yet.
- **Bottoms up test strategy** : implement integration testing by first integration test low level components and from there iterative integration test each ascending components and modules integration in the system. This ensures the programs module are integration tested early in the test phase, but post pone all high level components integration tests to late in the project

3.2 Design choice

Given the two different test options we went with a bottoms up test approach. This is due to how we have choose to implement the system which is a iterative bottoms up approach. Since we are striving for a Test Driven Development then the lower level components are always tested first. While we are black box testing the subsystem it becomes convenient to create a integration test while we are testing the system. This will lead to the more complicated components will be tested last which will be a challenge, but hopefully our extensive work with the lower level components will ease the challenge of testing the most complicated subsystems

4 System Testing

4.1 systemtesting

This system testing strategy guide has been written to ensure that the complete system atleast complies with the functional and nonfunctional requirements described in the RAD document.

4.2 Product, Revision and Overview

AutoSys is a dedicated provisioning service that provisions study related data to the client, based on task requests defined in the Study Configuration. The AutoSys Configuration Server is used to define the configuration of a given study for a team of researchers. The server stores study data and provides the client with the appropriate requested data. Also, the server delivers review tasks for the client used to conduct studies. Thus, the purpose of the system is to provide a tool support for conducting a study by a team of researchers
See RAD for more information.

Tabel 1: Planned releases

Version	**Release Date**	**Changes**
Initial Release (v1)	2015-12-10	
Revision 1	2015-12-18	

4.3 Features to be tested

The following features will be tested as a minimum to verify that the system conforms to the most important functional requirements:

- Create a team with a manager
- Create a Study Configuration
  - Assign a team
  - Define a phase
  - Create inclusion, exclusion and classification criteria
- Filter research papers using comparison operators
- Export papers in csv
- Handle multiple concurrent client requests
- Generate a research protocol

The following features will be tested as a minimum to verify that the system conforms to the most important nonfunctional requirements:

- The user must be able to configure a study using at most three application windows
- The system should restart within 30 seconds upon failures.
- The system should handle a client task request within 15 seconds.

4.4 Features not to be tested

- Platform compatibility on devices running other OS than Windows 10
- Database compatibility level Import and export of pdf files
- All features in the blue system

4.5 Environment Configuration

Tabel 2: Platform environment

	Macbook Pro (15-inch, 2015)	**Dell XPS 13**
Operating system	Windows 10 (64-bit) Bootcamp	Windows 10 (64-bit)
Memory	16 GB	8 GB
Hard disk	500 + GB, SSD	500 + GB, SSD
Processor	2.5GHz Intel Core i7-4870HQ	Intel Core i7-6500U
Graphics card	NVIDIA GeForce GT 750M	Intel HD Graphics 5200
Screen	15.4-inch 2,880 x 1,800 Retina screen	13.3? QHD+ (3200 x 1800)
Browser	Chrome	Edge
Network adapter	802.11ac wireless	100 MB Ethernet

4.6 System test methodology.

The system will primarily be tested from the functional requirements (from RAD) and the nonfunctional requirements (SDD). However, other activities such as Pilot testing, Acceptance testing or Installation testing might be used to further verify the system requirements. By way of example, we could select a group of potential users to test the common functionality described previously. The functional tests will be carried out by finding test cases from the use case models in the RAD document. The goal is to find differences between the functional requirements and the system.

4.7 Initial Test requirements

Test strategy (this document), written by test personnel, reviewed by product team, agreed to by project manager

4.8 System test entry and exit criteria

Entry Criteria
The software must meet the criteria below before the product can start system test.

- All basic functionality must work.
- All unit tests run without error.
- The code is frozen and contains complete functionality.
- The source code is checked into the Git repository.
- All code compiles and builds on the appropriate platforms.
- All known problems posted to Git.

Exit Criteria
The software must meet the criteria below before the product can exit from system test.

- All system tests executed (not passed, just executed).
- Results of executed tests must be discussed with product management team.
- Successful generation of executable images for all appropriate platforms.
- Code is completely frozen.
- Documentation review is complete.
- There are 0 showstopper bugs.
- There are fewer than $< x >$ major bugs, and $< y >$ minor bugs.

Test Deliverables

- Automated tests in $< framework >$
- Test Strategy and SQA (Software Quality Assurance) project plan
- Test procedure
- Test logs
- Bug-tracking system report of all issues raised during SQA process
- Test Coverage measurement

4.9 Test Cases

- Add test case for ExportProtocol use case
- Add test case for RetrieveStudyInformation use case
- Add test case for ManageStudy use case

5 Usability Testing

To get a better understanding on the user of the Autosys system and their usage of the "Study Configuration UI", one must perform usability tests to get their feedback. It is worth to mention the design goal: Ease of use where a study con figuration should be relatively easy to setup since it makes up the foundation that all study work processes rely on. (See System Design Document, Section 1.2 Design Goals).

At the initial design of the user interface various paper mocks up will be created and used to conduct scenario tests combined with think-aloud tests. The user will perform pseudo actions on the mocks while explaining the testing team what they are doing and why they are doing it. This enables the testing team to notice whether a given mock is living up to given usability tasks and if there are any usability defects that must be addressed. This will be conducted as an iterative process until some final mockups have been created.

A final mock will be chosen and will then be implemented. In some given point of time, a prototype will be created and used for conducting prototype tests. For this project, a horizontal user interface prototype will be used. This enables the test team to have a notion on how users interact with an actual interface and can thus address issues that may occur. The test team may want to test if the system is fit for use, ease of learning and task efficiency. Fit for use because it is important to check whether the user can perform all of his tasks. Ease of learning is to check how well the user is able to interact with the interface correctly and task efficiency is to check how well the user is performing his tasks. All of these usability measures are then taken into consideration and used to optimize the user interface.

When the system has most of its functionalities implemented, a product test will be conducted to test if the implemented functionalities are working as requested.