Test Semester Project
Group: Casper A. Jørgensen, Martin B. Rasmussen & Søren B. Sørensen

Pre-project

Project Introduction

For our semester project, we used the semester project from our database class as the base project.
This meant that our database project should meet it's own requirements, but also be able to meet the requirements to fulfill our learning goals.

So to introduce our project, let's start with our described learning goals:

- *Use unit testing as a way of testing our code.*
- *Mocking, to mock out our DB and test it without the DB.*
- *Parameterized testing for to test if the data fits with our expected results.*
- *Use of selenium to test our web application output.*
- *Continues integration to automatically run our test.*

Project planning
To best facilitate unit testing and mocking, we agreed to approach our entire project using TDD.
By writing tests before we implemented any of the user stories from our project, we ensure that the design of the code is testable and fulfills the requirements. It also ensures quick feedback - especially when bundled together with a CI tool that automatically runs tests every time our master branch (github) is updated.

Goals for base project (database)
In short, the application should be able to run 4 different queries - But it should be able to do so for 2 different types of databases. We chose Mysql and MongoDB to compare. It's basically about books, their respective authors and mentions of cities in the books.

In the following pages, we explain the actual use of the tools as well as reflect upon our experience with them.

Post-project

Unit Testing and Mocks
Firstly, because we adhered to TDD, we started with writing unit tests for all of our tasks. Obviously these would fail because no code was implemented.
However, besides facilitating the design of our system through failing tests, we used **mocks**.
Mocking out a facade-layer object, as well as our model classes, we ensured the design of each of the methods connecting to our mocks.
The unit test itself was made to ensure expected data (asserts) etc. However, mocks are also great for ensuring a specific behavior.

As our actual implementation began happening further into the project, we reiterated our unit tests to use the actual project.

Our unit tests are structurally very basic: *Arrange, Act, Assert.* Because our test project is built around our database project, a lot of the things we tested were focused on retrieving data from the databases. This meant:
Arrange focused on setting up a set of expected data and objects to work with.
Act carried out the functionality of the program, and retrieved the actual data from the database.
Assert, as is inferred by the name, verifies that the actual data is coherent with the expected data.

With a smaller dataset we verified that they were matching. However, as more and more books entered our library, more data was retrieved. So some tests were rewritten to check that the expected, *and possibly more*, were retrieved.

Parameterized Selenium Webdriver tests
Seeing as neither project (db or test) had any requirements for our UI, we decided to do an ASP.NET application (web). In part because it opened up the possibility for writing selenium tests.
- Selenium can be used in C# as well as Java and other applications, but we decided to do them in Java, as to very clearly underline that the selenium tests are ***absolutely decoupled*** from the source code of the application.
Because our application is supposed to retrieve data based on input (i.e. All books containing a specific city for example), we could greatly increase the effectivity of tests by parameterizing them. So instead of running our test agains *one* particular dataset, we would run it for 4-5 datasets and all expected results would be checked.

Full disclosure; Our selenium tests work great, and we even got our CI tool to run them after building the application. But for some reason it fails on the CI builds after opening the webdriver, both chrome and firefox(gecko). Trying to access any webpage fails.

Continuous Integration tool[1]

For our CI platform, we chose to use Visual Studio Online (TFS).
During most of our development, every time we pushed any changes to the master branch our project was built and all unit tests were run.

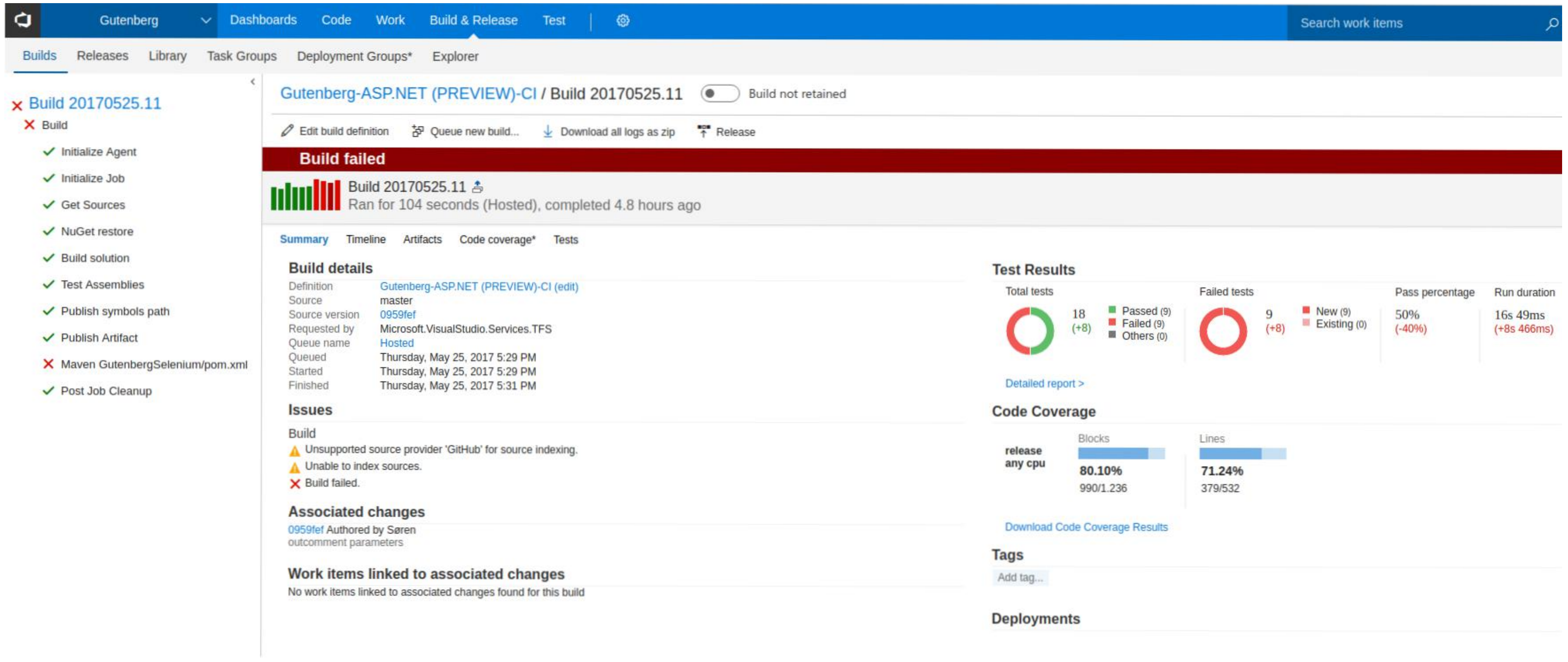
It wasn't till the later part of the project that our selenium tests were written and added to the project. TDD was definitely revolving around our unit tests, and not our tests of the finished system.
This follows Mike Cohn's testing pyramid quite well; Most unit tests, less UI tests.
In principle it's because a defect in the UI could be because of many different parts of the system. Unit tests however test 1 specific part of the program and gives result (one cog in the machine).

We activated Code Coverage in our CI tool for every test run. This wasn't part of our learning goals, but it *did* provide insight into how well we tested our different methods. The code coverage from VS Online is quite rough, it's mostly quantitive (percentage of blocks covered).

Our CI tool was a really great tool during our development. The feedback was automatic and very fast. It basically meant that after pushing and merging, before we knew it everything was tested and a detailed report was written.
Even better, we were able to bundle together unit tests and selenium tests in one easy environment.

Even on our PC's we had to run the development environment for selenium to test it!

[1] See appendix for screenshot of result from a build. The build failed because selenium tests couldn't open drivers - Worked fine on our laptops



Build 20170525.11
- ✗ Build
  - ✓ Initialize Agent
  - ✓ Initialize Job
  - ✓ Get Sources
  - ✓ NuGet restore
  - ✓ Build solution
  - ✓ Test Assemblies
  - ✓ Publish symbols path
  - ✓ Publish Artifact
  - ✗ Maven GutenbergSelenium/pom.xml
  - ✓ Post Job Cleanup

Gutenberg-ASP.NET (PREVIEW)-CI / Build 20170525.11 — Build not retained

Build failed

Build 20170525.11
Ran for 104 seconds (Hosted), completed 4.8 hours ago

Build details

Definition	Gutenberg-ASP.NET (PREVIEW)-CI (edit)
Source	master
Source version	0959fef
Requested by	Microsoft.VisualStudio.Services.TFS
Queue name	Hosted
Queued	Thursday, May 25, 2017 5:29 PM
Started	Thursday, May 25, 2017 5:29 PM
Finished	Thursday, May 25, 2017 5:31 PM

Issues

Build
- ⚠ Unsupported source provider 'GitHub' for source indexing.
- ⚠ Unable to index sources.
- ✗ Build failed.

Associated changes

0959fef Authored by Søren
outcomment parameters

Work items linked to associated changes

No work items linked to associated changes found for this build

Test Results

Total tests	Failed tests	Pass percentage	Run duration
18 (+8) — 🟩 Passed (9), 🟥 Failed (9), ■ Others (0)	9 (+8) — 🟥 New (9), 🟥 Existing (0)	50% (-40%)	16s 49ms (+8s 466ms)

Detailed report >

Code Coverage

	Blocks	Lines
release any cpu	80.10% 990/1.236	71.24% 379/532

Download Code Coverage Results

Tags

Add tag...

Deployments