AFSTUDEERVERSLAG

Kwaliteitsverbetering door test automation

Door Dirk Luijk • 9 januari 2014



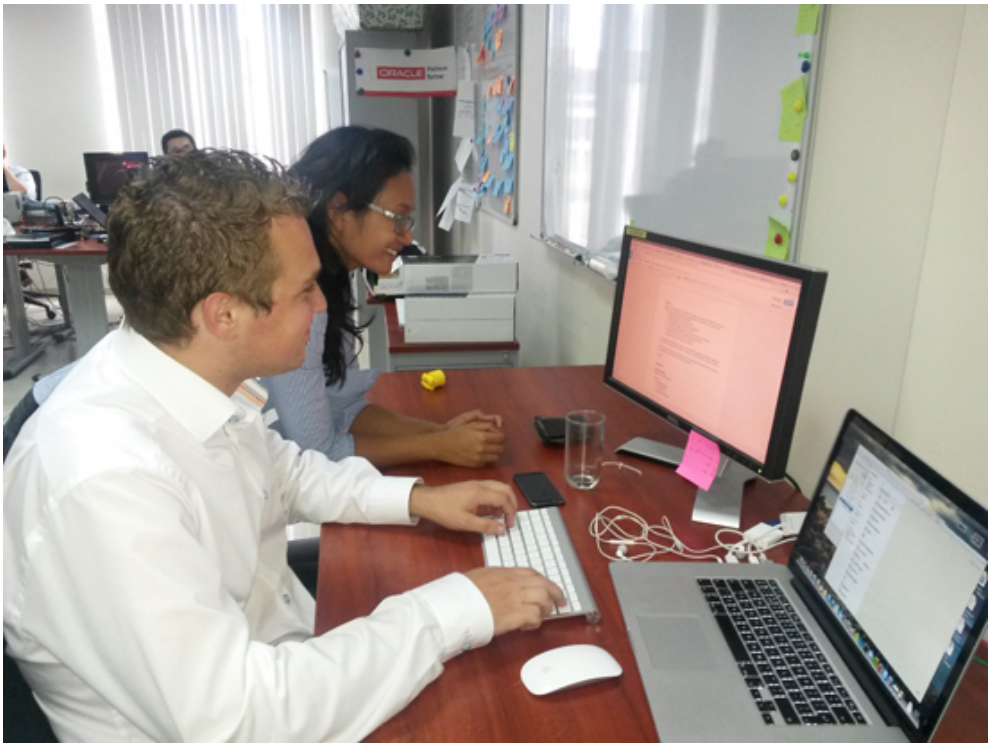
Student	**Dirk Luijk**
Afstudeerperiode	**18 augustus t/m 19 december 2014**
Inleverdatum	**vrijdag 9 januari 2015**
School	**De Haagse Hogeschool, hoofdvestiging**
Opleiding	**Informatica (voltijd)**
Studentnummer	**10026312**
Begeleidend examinator	**mw. A. Lousberg**
Tweede examinator	**mw. G.E. in 't Veld**
Studieloopbaanbegeleider	**mw. A. Wieman**
Bedrijf	**Qualogy Suriname**
Opdrachtgever	**dhr. P. Sweeb**
Bedrijfsmentor	**dhr. R. Quartier**

Referaat

D. Luijk, 10026312, Afstudeerverslag, Kwaliteitsverbetering door test automation bij Qualogy Suriname, 9 januari 2015. Persoonlijk procesverslag van het project "Kwalteitsverbetering door test automation", uitgevoerd in het kader van het afstudeertraject bij de Haagse Hogeschool, opleiding Informatica, voltijd.

Trefwoorden

- Test automation
- Continuous integration
- Web testing
- Interview
- Literatuuronderzoek



Voorwoord

Het document wat voor u ligt vormt het verslag van mijn afstudeerperiode bij Qualogy in Suriname, een periode waarin ik veel nieuwe ervaringen heb kunnen opdoen op veel verschillende vlakken.

Allereerst was het een bijzondere ervaring om in een vreemd land aan de andere kant van de wereld af te studeren. Hoewel het bedrijf zelf redelijk Nederlands aanvoelde, heb ik tijdens de vier maanden dat ik Suriname verbleef op sociaal en cultureel vlak veel nieuwe indrukken opgedaan. Suriname is een prachtig land met een aangenaam klimaat, lekker eten en gezellige mensen. Naast de opgedane werkervaring heb ik ruimschoots kunnen genieten en mijn horizon kunnen verbreden.

Binnen Qualogy is er veel diversiteit en omdat de scope van mijn opdracht meerdere afdelingen besloeg, heb ik veel van het bedrijf mogen zien. Zo heb ik met management, testers en Java- en Oracle ontwikkelaars mogen samenwerken om het gehele proces van kwaliteitsmanagement te kunnen verbeteren.

Daarnaast was testen voor mij een nog onbekend vakgebied. Ik kende het wel uit de theorie die op school is gegeven, maar in de praktijk heb ik er nog weinig mee gedaan. Daardoor was het absoluut een uitdaging om in dit voor mij nieuwe vakgebied binnen een onbekend bedrijf in een vreemd land advies te moeten gaan geven voor het verbeteren en automatiseren van het testproces. Ik heb dan ook door middel van zelfstudie veel nieuwe kennis moeten opdoen om de opdracht te kunnen uitvoeren. Implementatie en oplevering was echter minstens zo belangrijk, waardoor ik ook niet te lang bij de theorie kon blijven hangen.

De opdracht is uitgevoerd in samenwerking met mijn manager, Purcy Sweeb, en mijn bedrijfsmentor, Ray Quartier. Gezamenlijk hebben we de onderzoeksvragen bedacht en de scope van de opdracht bepaald. Ook hebben zij mij begeleid tijdens het uitvoeren van de opdracht. Zij waren nooit te beroerd om mijn vragen te beantwoorden. Bij deze wil ik hen daarom bedanken voor hun ondersteuning tijdens het traject. Verder wil ik ook mijn begeleiders vanuit de Haagse Hogeschool bedanken voor hun begeleiding. Hun waardevolle feedback heeft geleid tot nieuwe inzichten en heeft gezorgd voor een completer verslag.

Tenslotte wil ik ook de collega's bij Qualogy bedanken voor de fijne samenwerking. Voorafgaand aan de afstudeerperiode hield ik rekening met te weinig draagvlak voor de veranderingen die doorgevoerd moesten worden, maar uiteindelijk heb ik met weinig weerstand te maken gehad waardoor ik mijn opdracht succesvol heb kunnen uitvoeren.

Ik wens u veel leesplezier toe.

Dirk Luijk
Alblasserdam, 9 januari 2015

Inhoudsopgave

Inleiding 6
1. Situatiebeschrijving 7
1.1 Over Qualogy Suriname 7
1.2 Positionering 8
1.3 Probleemstelling 8
1.4 Doelstelling 8
2. Voorbereiding 9
2.1 Voorafgaand aan het traject 9
2.2 Eerste week: korte inventarisatie 9
2.2 Plan van aanpak 9
3. Onderzoeksfase 11
3.1 Inventarisatie bestaande situatie 11
3.2 Proefdraaien met Selenium IDE 14
3.3 Voorlopig advies 16
3.4 Literatuurstudie 18
3.5 Toolselectie 22
3.6 Adviesrapport 27
4. Implementatiefase 29
4.1 Voorbereiding 29
4.2 Continuous integration 32
4.3 Unit testing 34
4.4 Browsertesten (web testing) 36
5. Oplevering 40
5.1 Overdracht aan collega 40
5.2 Presentatie afstudeeropdracht 40
6. Evaluatie 42
6.1 Keuze afstudeeropdracht- en bedrijf 42
6.2 Plan van aanpak 42
6.3 Onderzoeksfase 43
6.4 Implementatiefase 43
6.5 Resultaten & oplevering 44
6.5 Begeleiding 45
6.6 Beroepstaken 45
6.7 Conclusie 47
Literatuurlijst 48
Internetbronnen 48
Literatuur 48
Verklarende woordenlijst 49
Bijlage A - Plan van aanpak 51
Bijlage B - Interviews 52
Bijlage C - Beschrijving bestaande situatie 53
Bijlage D - Conceptadvies 10 september 54
Bijlage E - Toolselectierapport 55
Bijlage F - Adviesrapport 56
Bijlage G - Evaluatieformulier 57

Inleiding

Dit verslag beschrijft op welke wijze mijn afstudeeropdracht bij Qualogy is uitgevoerd.
Deze afstudeer- opdracht vond plaats ter afsluiting van mijn opleiding Informatica van De Haagse Hogeschool. Het verslag is primair beschreven om de examinatoren inzicht te geven in het verloop van het traject. De manier van schrijven is echter zodanig dat een ieder die niet betrokken is geweest bij de opdracht kan begrijpen hoe de opdracht is uitgevoerd en waarom bepaalde beslissingen gemaakt zijn.

Het verslag beschrijft allereerst de voorbereiding op het traject, die nodig waren om tot een plan van aanpak te kunnen komen. Daarna volgt de onderzoeksfase, waarin geprobeerd wordt om tot een passend advies te komen. Deze fase nam ongeveer de helft van de tijd in beslag. Daarna volgt de implementatiefase en de afronding. Van elke fase wordt verslag gedaan van de verrichte werkzaamheden, de opgeleverde producten en de resultaten. Ook worden knelpunten geëvalueerd. De producten die opgeleverd zijn aan Qualogy zijn als bijlage opgenomen achterin in dit afstudeerverslag.

1. Situatiebeschrijving

In dit hoofdstuk wordt de situatie beschreven zoals deze eruit zag bij Qualogy ten tijde van de aanvang van de opdracht. Tevens wordt de probleem- en doelstelling vastgesteld.

1.1 Over Qualogy Suriname

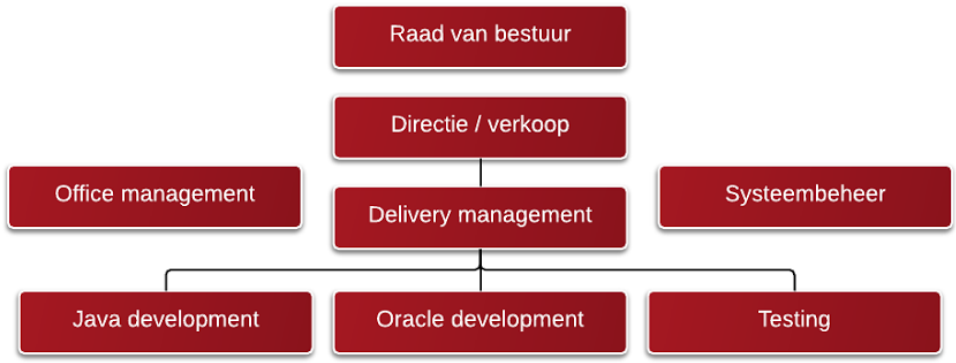
Qualogy is een Nederlands bedrijf, opgericht in 1998 en gevestigd te Rijswijk. Het levert maatwerk in Oracle- en Java-applicatieontwikkeling, Oracle E-Business, database- en applicatiebeheer, Business Intelligence, Agile Consultancy en Web 2.0 development. Onder andere AHOLD, ArboNED BV, C1000, Canon Europa NV, Gemeente Den Haag en Tele2 Nederland behoren tot de klantenkring van Qualogy.

Qualogy Suriname is in 2007 opgericht en is gevestigd te Paramaribo. Inmiddels opereert Qualogy Suriname zelfstandig en heeft zij haar eigen klantenkring in Suriname en Caribisch gebied. Een sterk punt is dat Qualogy Suriname haar eigen personeel opleidt tot young professionals. Er zijn op dit moment ongeveer 40 mensen werkzaam.

Bij Qualogy Suriname is er vooral sprake van een lijn-staforganisatie, maar er zijn ook kenmerken van een matrixorganisatie:

- Personen die hetzelfde werk doen, behoren tot dezelfde afdeling. Zo is er een testafdeling en aparte ontwikkelafdelingen voor Java en voor Oracle. Elke afdeling bevat een manager (lead developer, testmanager) aan wie gerapporteerd wordt.
- Personen die aan hetzelfde project werken, behoren tot een projectgroep en rapporteren naast de manager van de afdeling ook aan de projectmanager. Bij Qualogy is dit momenteel Purcy Sweeb. De Java-teams zitten in aparte ruimtes, waar ook de stand-up meetings plaatsvinden.

De organisatiestructuur is zichtbaar in figuur 1.1.



Figuur 1.1 - Organogram Qualogy Suriname

1.2 Positionering

Mijn positie binnen Qualogy is niet beperkt geweest tot een specifieke afdeling. In het begin van het afstudeerproject was ik werkzaam bij Quality Control (de testafdeling), waar drie testers zaten. Purcy Sweeb, delivery manager, werkte in diezelfde ruimte. Later verhuisde ik samen met de testafdeling naar de Oracle-kamer en zat ik tussen de Oracle-ontwikkelaars. Ik werkte daar voornamelijk zelfstandig. Tijdens de implementatiefase heb ik op de Java-ontwikkelafdeling gewerkt, binnen het team van het KvK-project.

1.3 Probleemstelling

De probleemstelling is tweeledig:

- Het testen van software kost op dit moment teveel tijd. Bovendien is Qualogy aan het overstappen op Agile ontwikkelmethoden zoals Scrum, waarbij de veel voorkomende herhaling (iteraties) ervoor zorgen dat er meer en vaker getest moet worden. Dit vraagt om een hogere efficiëntie van het testproces. Het betreffen hier vooral functionele (regressie)testen.
- Op dit moment is de door het ontwikkelteam opgeleverde software (system under test) vaak van onvoldoende kwaliteit, waardoor er onnodig veel getest moet worden voor het systeem aan de afgesproken kwaliteit voldoet. Kwaliteitsverbetering van het testproduct is nodig. Unittests worden nauwelijks toegepast.

1.4 Doelstelling

De afstudeeropdracht bestaat uit het verbeteren van het gehele testproces bij Qualogy Suriname, door middel van onderzoek, toolselectie, implementatie en training.

- Om tijd te besparen bij het testen en aan te sluiten bij Agile ontwikkelmethodes zoals scrum, wordt *test automation* ingevoerd. Ofwel: het testen wordt zoveel als mogelijk geautomatiseerd. Hiervoor moeten tools worden ingezet.
- Om de kwaliteit van het opgeleverde systeem (*system under test*) te verbeteren, worden automatische unittests ingevoerd en indien nodig *continuous integration*[1] toegepast, zodat de kwaliteit van het systeem continu bekeken kan worden.

Na het uitvoeren van het project moet de totaalduur van projecten korter zijn, door een efficiënte manier van automatisch testen. Verder moeten werknemers zelfstandig volgens de nieuwe methode en met de nieuwe tools kunnen werken.

[1] Zie de verklarende woordenlijst achterin dit verslag voor een definitie.

2. Voorbereiding

In dit hoofdstuk wordt verslag gedaan van de voorbereidingen die gedaan zijn voor het uitvoeren van de opdracht. Hieronder vallen zowel de voorbereidingen die gedaan zijn voorafgaand aan het traject als de voorbereidingen tijdens de eerste weken, zoals het opstellen van een plan van aanpak.

2.1 Voorafgaand aan het traject

Als belangrijkste voorbereiding op de afstudeeropdracht heb ik me verdiept in TMap Next en een aantal boeken hiervan aangeschaft. Ook heb ik mij verdiept in de Surinaamse cultuur om een beeld te krijgen van de gang van zaken op de werkvloer en hoe je het snelst kunt integreren binnen een dergelijk bedrijf. Verder heb ik ervoor gezorgd dat mijn eigen laptop goed was ingericht en naast Mac OS ook Windows draaide; zodat ik voor de toolselectie ook met tools voor Windows kon werken.

2.2 Eerste week: korte inventarisatie

Voordat ik een plan van aanpak kon opstellen, wilde ik eerst hetgeen in het afstudeerplan stond, verifiëren. Ook had ik nog te weinig inzicht in de bestaande situatie. Daarom heb ik een korte inventarisatie gedaan, waarbij ik bij de volgende zaken heb stilgestaan:

- Hoe zien de afdelingen eruit / hoeveel projecten lopen er?
- Hoe wordt er samengewerkt tussen ontwikkelaars en testers?
- Welke ontwikkel- en projectmanagementmethodes worden toegepast?
- Hoe staat het met unit testing?

Uiteraard hoefde ik nog niet alles direct te weten, het onderzoeken van de bestaande situatie was namelijk ook een onderdeel van de afstudeeropdracht. Voor het plan van aanpak was het echter wel handig om in ieder geval een globaal overzicht te hebben van het bedrijf en genoemde zaken.

De inventarisatie bestond voornamelijk uit het interviewen van drie collega's op de testafdeling en mijn manager. Dit interview is direct op de eerste werkdag (18 augustus) gehouden. Later volgde er nog meer interviews, maar voor het plan van aanpak was het eerste interview voldoende.

2.2 Plan van aanpak

Samen met de reeds eerder vastgestelde probleemstelling, doelstelling en planning, zoals beschreven in het afstudeerplan, was er nu voldoende informatie om een concreet plan van aanpak op te stellen voor het bedrijf.

Het plan van aanpak bestaat uit een probleem- en doelstelling, een beschrijving van de op te leveren producten, planning, risico's en randvoorwaarden. De probleem- en doelstelling komt nagenoeg overeen met die uit het afstudeerplan. Wat betreft de scope van de opdracht is het duidelijk geworden dat de opdracht niet alleen op de testafdeling zou plaats vinden, maar ook op de ontwikkelafdelingen van Qualogy. Wel is er in het plan van aanpak nog veel flexibiliteit aanwezig, omdat de onderzoeksfase bepalend zou zijn voor de implementatiefase.

Ik heb geprobeerd de doelstelling zo concreet en duidelijk mogelijk te beschrijven, ook wel SMART genoemd:

Specifiek	Binnen de test- en ontwikkelafdelingen van Qualogy Suriname worden de ontwikkel- en systeemtesten geautomatiseerd door middel van het integreren van tools en het aanpassen van de werkwijze, om daarmee tijd (en dus geld) besparen en de arbeidsvreugde van zowel ontwikkelaars als testers te verhogen.
Meetbaar	Bij het afronden van de opdracht moet er bij minimaal één lopend project al gebruik worden gemaakt met de tools, zodat de resultaten zichtbaar worden. Wat betreft de hoeveelheid tijd die bespaard wordt, is er geen richtlijn vastgelegd. Elke tijdswinst is meegenomen.
Acceptabel (haalbaar)	Er is een ruime planning, de resources zijn aanwezig en de risico's zijn in kaart gebracht.
Relevant	Het management heeft baat bij de kostenreductie en de kortere doorlooptijd van projecten. Ontwikkelaars kunnen op een eerder moment fouten oplossen en zullen meer inzicht hebben in de kwaliteit alvorens de systeemtesten. De testers zullen minder repeterend werk te hoeven verrichten.
Tijdgebonden	De opdracht wordt binnen 4 maanden afgerond. Er wordt een planning gemaakt met genoeg bufferruimte.

Wat betreft de uit te voeren werkzaamheden zijn er de volgende activiteiten gepland:

1. **Onderzoeksfase**
    - **Inventarisarie van de bestaande situatie**: allereerst wordt er onderzoek gedaan naar de bestaande situatie om de achtergrond van de genoemde problemen duidelijk te maken, de huidige werkwijze in kaart te brengen en een meetpunt te maken zodat later duidelijk gereflecteerd kan worden.
    - **Literatuurstudie:** er zal informatie worden verzameld over best practices voor test automation, zoals *continuous integration* en *browser automation*.
    - **Toolselectie:** er zal een toolselectie plaatsvinden, waarbij gezocht wordt naar passende tools waarmee men binnen Qualogy testen kan automatiseren, rekening houdend met de wensen van de opdrachtgever en de situatie binnen de organisatie.
    - **Adviesrapport opstellen:** in het adviesrapport worden alle bevindingen, zoals de resultaten van de toolselectie, gerapporteerd. Ook wordt er een advies uitgebracht en wordt aangegeven hoe deze geïmplementeerd kan worden binnen Qualogy.
2. **Implementatiefase**
   Na goedkeuring op het adviesrapport zullen de wijzigingen geïmplementeerd worden, waarbij de tools geïnstalleerd en geconfigureerd worden. Ook zullen er trainingen gegeven worden om de gebruikers te leren werken met de nieuwe software en met de nieuwe methodes.

3. Onderzoeksfase

In dit hoofdstuk komt het onderzoek aan bod dat is uitgevoerd binnen Qualogy Suriname. Het onderzoek vormt de basis voor het adviesrapport, de implementatie van de tools en de introductie van de nieuwe methodes.

Deze fase heeft ongeveer 60% van de tijd in beslag genomen. Dit was langer dan gepland; in paragraaf 3.6.4 leest u meer hierover.

3.1 Inventarisatie bestaande situatie

Allereerst is er een inventarisatie geweest om de bestaande situatie in kaart te brengen. Hier was ik al mee begonnen vóór het plan van aanpak, maar om alle details naar boven te krijgen heb ik in week 2 en 3 nog enkele interviews gehouden. Via mijn manager is er geïnformeerd naar geschikte werknemers die het meest afweten van de gang van zaken op de verschillende afdelingen. Vervolgens is er per e-mail een vragenlijst opgestuurd naar de deelnemers ter voorbereiding. De interviews zijn na afloop in eigen bewoording uitgewerkt en ter controle opgestuurd naar de deelnemers. Geprobeerd is om alle informatie uit de interviews vast te leggen, voor zover dit mogelijk is.

De interviews bevatten o.a. de volgende vragen:

- Hoeveel mensen werken er op de afdeling?
- Hoeveel projecten worden er hier tegelijk uitgevoerd?
- Welke ontwikkelmethode gebruiken jullie?
- Hoe ziet de documentatie van een project eruit?
- Op welke manier leggen jullie requirements vast?
- Maken jullie gebruik van unittests?
- Zo ja, Hoe groot denk je dat de dekkingsgraad van jullie unittests is?
- Maken jullie gebruik van een versiebeheersysteem?

Op deze manier ben ik te weten gekomen hoe men op de verschillende afdelingen werkt, hoe de projecten en teams eruit zien, welke methodes worden gebruikt en tenslotte ook hoe het gesteld is met zaken als versiebeheer en unittesting.

In de volgende paragrafen worden de uitkomsten van de inventarisatie beschreven.

3.1.1 Quality Control

Op de testafdeling werd vroeger volgens TMap NEXT gewerkt, dit bleek echter in de praktijk niet goed te passen bij de situatie binnen Qualogy Suriname omdat dit teveel papierwerk opleverde. De projecten zijn volgens de testers te klein om bijvoorbeeld met een master- en detailtestplan te werken. Bovendien wordt er niet expliciet risico-gedreven getest zoals TMap voorschrijft. Daarom is men overgestapt op Rapid Software Testing. Daarbij maakt men gebruik van mindmaps in combinatie met exploratory testing.

Tijdens het testen worden alle stappen 'opgenomen' met RapidReporter. De gevonden issues worden vervolgens gerapporteerd via JIRA, een platform voor projectmanagement dat ook door de ontwikkelaars gebruikt wordt. Er is dus nog nauwelijks sprake van enige automatisering.

Acceptatietesten, die aangeven of het systeem in gebruik genomen kan worden, worden in feite min- of meer door de afdeling Quality Control uitgevoerd. Wanneer alle functionaliteiten correct werken in overeenstemming met het functioneel ontwerp, wordt de klant geïnformeerd. Deze kan het vervolgens zelf nog uitproberen alvorens het in productie wordt genomen.

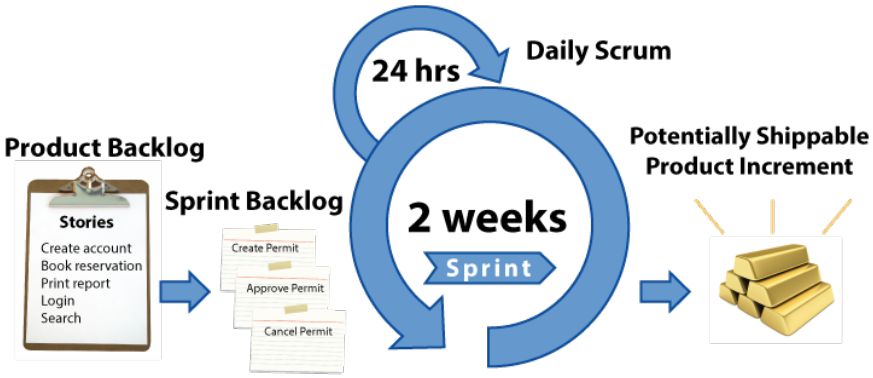
De afdeling bestaat uit vier personen, waarvan één testmanager is. De tests zijn niet risico-gedreven, bij het maken van de planning en het bepalen van het testniveau is de beschikbare tijd leidend. Meestal wordt ongeveer een kwart van de ontwikkeltijd aangehouden voor het bepalen van de hoeveelheid testuren.

3.1.2 Java-afdeling

De afdeling Java bestaat uit vier Scrum teams. Op advies van mijn opdrachtgever is er eerst een interview geweest met Jozua Herfst van het KvK-project. Dit project betreft het ontwikkelen van een webapplicatie voor de Kamer van Koophandel op Aruba, waarbij alle formulieren die vroeger handmatig ingevuld moesten worden, nu online ingevuld kunnen worden. Vanuit een backoffice kan men bij de Kamer van Koophandel deze formulieren verwerken.

Uit dit interview bleek dat een ander projectteam, namelijk die van het UTS-project, al met Jenkins werkt. Jenkins is een tool voor continuous integration[2]. Daarom is besloten om nog een interview te houden met Rogér Pique en Ray Amatsoleman van het UTS-team, om te vragen naar hun ervaring met Jenkins. Hieruit bleek dat zij het al hadden draaien inclusief deployment. Uit de interviews bleek ook dat het kennisniveau en de mate van kwaliteitsmanagement tussen de verschillende Java-teams verschilt. Zo past het UTS team volledige *continuous integration* toe m.b.v. Jenkins en werken ze veel meer met unit tests, terwijl dit bij andere teams niet het geval is. Harde cijfers m.b.t. de dekkingsgraad van de unittests waren echter niet bekend. De dekkingsgraad is een manier waarmee je kunt aangeven in hoeverre de code van een applicatie gedekt wordt door unit tests. Een dekkingsgraad van 80% is bijvoorbeeld beter dan een dekkingsgraad van 10%.

De Java-teams werken allen volgens scrum. Scrum behoort tot de Agile methodes en is handig bij projecten waarbij de klant nog niet goed weet wat hij wil. Scrum heeft de flexibiliteit om met laat wijzigende eisen en wensen om te gaan. Er wordt gewerkt in multidisciplinaire teams die in korte sprints een werkend product moeten opleveren.



Figuur 3.1 - Scrum werkt met korte sprints waarbij er een werkend product wordt opgeleverd.

[2] Zie de verklarende woordenlijst achterin dit verslag voor een definitie.

De Java-teams werken allen volgens scrum met sprints van 1 of 2 weken en gebruiken JIRA Agile voor het beheren van de backlog met user stories. JIRA Agile is een webapplicatie ontwikkeld door Atlassian, geschikt voor het projectmanagement van softwareprojecten.

Verder gebruikt men JUnit voor unittesting, Maven als build-tool en Subversion voor versiebeheer. Het UTS-team maakt als enige team gebruik van Jenkins voor *continuous integration* en deployment.

3.1.3 Oracle-afdeling

Bij de Oracle-afdeling is er een interview gehouden met Sullivan Kromosoeto, Oracle Lead Developer. Op deze afdeling wordt er gewerkt met een watervalmethode. Dit houdt in dat de documentatie vaak heel uitgebreid is, omdat men pas gaat bouwen wanneer het ontwerp gereed en goedgekeurd is. Men werkt in kleine groepen (2 á 3 man) of soms zelfstandig aan projecten. Er worden geen unittesten gebruikt, hoewel er bij een gedeelte van het team wel al draagvlak voor is.

Op de Oracle-afdeling werken er twee soorten ontwikkelaars:

- **PL-SQL developers**: zij bouwen de database, met bijbehorende business layer. Alle logica, zoals business rules, constraints en functies, komen hierin terecht.
- **Oracle Apex developers**: zij bouwen de applicatie. Dit bestaat uit een tamelijk 'dunne' frontend laag op de Oracle database. Deze applicatie wordt gebouwd met Oracle Application Express, ook wel *Apex* genoemd.

Voor versiebeheer wordt er niet gebruik gemaakt van een bepaalde tool. Men laat de afdeling DBA (*database administration*, systeembeheer) elke dag backups maken van de code. Ook wordt er geen *continuous integration* toegepast. Sullivan (Oracle developer) denkt dat dit niet nodig is, aangezien de projectgroepen klein zijn en er vrijwel nooit conflicten zijn. Men werkt bovendien gezamenlijk in dezelfde ontwikkelomgeving.

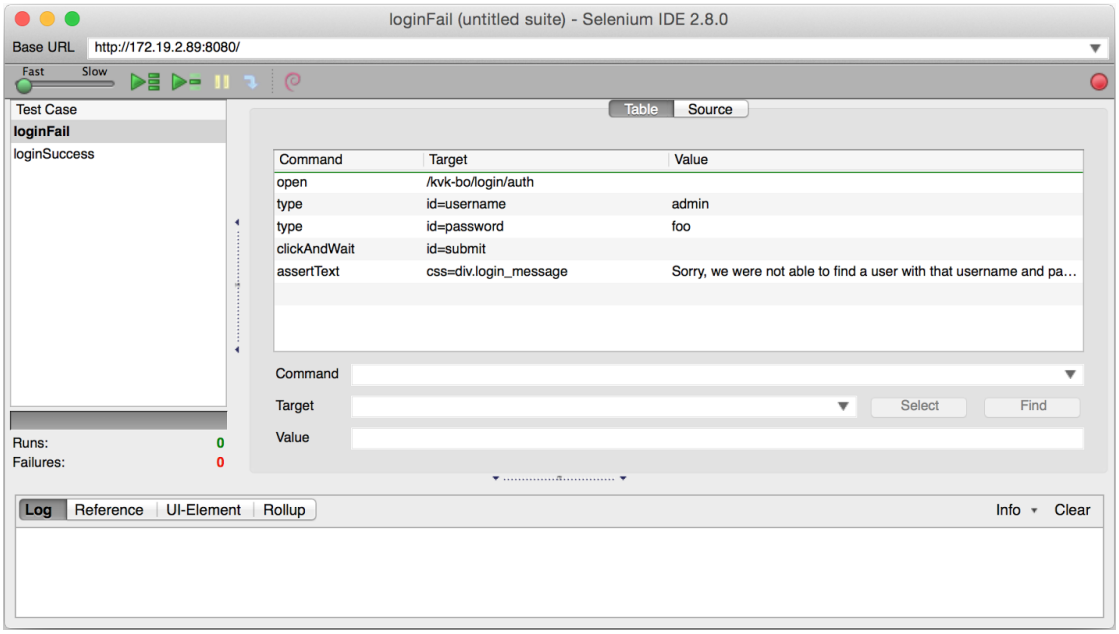
3.1.4 Samenvatting

Op basis van deze eerste inventarisatie kunnen de volgende conclusies al getrokken worden:

- De testafdeling werkt zelfstandig en het testen gebeurt handmatig en op een tamelijk informele wijze.
- De afdeling Java bestaat uit losse en van elkaar verschillende teams, in tegenstelling tot de Oracle-afdeling. Het zou goed zijn als er meer kennisuitwisseling plaats vindt zodat men van elkaar kan leren en men op een uniforme wijze blijft werken.
- Vanwege de karakteristieken van scrum (veranderlijke requirements) is het aan te raden om unit testing meer toe te passen bij de Java-teams. Continuous integration kan helpen om zaken als dekkingsgraad en het verloop van een project inzichtelijk te maken.
- De Oracle-afdeling werkt volgens een watervalmethode en aangezien de requirements niet wijzigen tijdens het traject, zijn er veel minder testsessies. Continuous integration lijkt niet nodig te zijn.

3.2 Proefdraaien met Selenium IDE

In week 3 zijn we, zonder de toolselectie af te wachten, alvast gaan werken met Selenium. De belangrijkste reden hiervoor is dat er een snelle oplossing nodig was voor het automatiseren van de testen voor het KvK-project, omdat men daar veel tijd aan kwijt was. Bovendien is Selenium een gratis tool en eenvoudig te leren. Het KvK-project is een webapplicatie voor de Kamer van Koophandel van Aruba waarbij men online de formulieren kan invullen. Vanuit een backoffice kunnen deze formulieren vervolgens verwerkt worden.



Figuur 3.2 - Het hoofdscherm van Selenium IDE.

Selenium IDE is een zogenaamde *record- en playback* tool. Dit houdt in, dat je de testscripts kunt opnemen en deze later opnieuw kan afspelen. De IDE is beschikbaar als plugin voor Mozilla Firefox. Het programma kent een eenvoudige interface, met de volgende onderdelen en functionaliteiten:

- In het hoofdscherm is het huidige testscript te zien. Een testscript bestaat uit *commands*, met elk een *target* en een *value*.
    - Er zijn meerdere soorten *commands*: uitvoerende opdrachten, bijvoorbeeld het openen van een pagina, het typen in een bepaald veld of het klikken op een link. Verder heb je nog *wait*-opdrachten, die nodig zijn wanneer er bijvoorbeeld een AJAX-request op de achtergrond wordt uitgevoerd. Tenslotte zijn er ook nog de *verify* en *assert*-opdrachten. Hiermee kunnen er controle's uitgevoerd worden: de daadwerkelijke tests. Het verschil tussen een verify en een *assert* is dat het script afgebroken wordt bij een falende *assert*, niet bij een falende *verify*.
- Links is een overzicht met de test cases. Deze vormen samen een test suite.
- Bovenaan staan knoppen voor het opnemen en afspelen. De snelheid van het afspelen kan ingesteld worden. Ook is het mogelijk om breakpoints te plaatsen of het script te pauzeren.

Verder is het mogelijk om het testscript te exporteren naar vele verschillende programmeertalen, waaronder Ruby (RSpec), Python, Java (JUnit 3, JUnit 4, TestNG) en C# (NUnit). De gegenereerde code is bruikbaar, maar behoeft wel enige refactoring, zoals het verwijderen van dubbele en ongebruikte code.

```java
/**
 * @author Dirk Luijk <dluijk@qualogy.sr>
 */

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;

import java.util.concurrent.TimeUnit;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.fail;

public class LoginFail {
    private WebDriver driver;
    private String baseUrl;
    private boolean acceptNextAlert = true;
    private StringBuffer verificationErrors = new StringBuffer();

    @Before
    public void setUp() throws Exception {
        driver = new FirefoxDriver();
        baseUrl = "http://172.19.2.89:8080/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    }

    @Test
    public void testLoginFail() throws Exception {
        driver.get(baseUrl + "/kvk-bo/login/auth");
        driver.findElement(By.id("username")).clear();
        driver.findElement(By.id("username")).sendKeys("admin");
        driver.findElement(By.id("password")).clear();
        driver.findElement(By.id("password")).sendKeys("foo");
        driver.findElement(By.id("submit")).click();
        assertEquals("Sorry, we were not able to find a user with that username and passwo
    }
```

Figuur 3.3 - Selenium IDE kan een testscript converteren naar o.a. Java-code.

Het proefdraaien van Selenium was niet ingepland, maar de planning was redelijk ruim genomen en dus kon het tussen de werkzaamheden door uitgevoerd worden. Bovendien vormde het in feite al een stuk van de toolselectie. Al zou er na de toolselectie een betere tool naar voren komen, dan kon Selenium altijd nog vervangen worden.

Tijdens het proefdraaien hebben we een gedeelte van de testgevallen van de frontoffice van KvK door middel van Selenium IDE kunnen automatiseren. De testafdeling kon hier zelfstandig mee verder werken en na een aantal weken waren alle testgevallen geautomatiseerd. Dit heeft veel tijdwinst opgeleverd. Ik heb mijn mentor gevraagd om hiervan een inschatting te geven. Dit is terug te vinden in paragraaf 6.5.

De ervaringen met Selenium hielpen mee om de randvoorwaarden voor de toolselectie te bepalen. Tevens werden de voor- en nadelen van record-and-playback tools dankzij het proefdraaien beter zichtbaar. In paragraaf 3.5 worden er een aantal van deze voor- en nadelen genoemd.

3.3 Voorlopig advies

Aangezien mijn manager in september op vakantie ging, werd mij gevraagd om vóór zijn vertrek al met een voorlopig advies te komen. Op 10 september is er daarom al een kort document van twee pagina's opgeleverd waarin de eerste bevindingen van de interviews en de proefsessie met Selenium IDE werden beschreven.

3.3.1 Bevindingen proefsessie Selenium

Men had op dat moment bij de testafdeling ongeveer anderhalve week gewerkt met Selenium op het KvK-project en er waren al resultaten geboekt. Zowel Kim, Gail als Ray konden ermee werken en er werd daadwerkelijk tijdswinst behaald.

Een probleem was echter nog wel dat er weinig formele eisen waren voor de te gebruiken tool. De bevindingen die ik had geconstateerd bij het proefdraaien met Selenium waren gebaseerd op de volgende, voornamelijk zelf bedachte criteria:

1. De tool moet gratis zijn of weinig kosten
2. De testers moeten er onafhankelijk (zonder hulp van een programmeur) mee kunnen werken
3. De testscripts moeten weinig onderhoud vereisen bij veranderingen

Het eerste criterium was al een keer genoemd door de opdrachtgever. Het tweede criterium heb ik zelf opgesteld en leek op dat moment ook vanzelfsprekend: de testers werken namelijk in de bestaande situatie los van de ontwikkelaars en hebben bovendien weinig programmeervaardigheden. Het derde en laatste criterium is gebaseerd op het idee dat er met scrum gewerkt wordt en er dus veranderingen kunnen optreden tussen de sprints. Wanneer er teveel onderhoud nodig is aan de testscripts, heeft automatiseren geen zin omdat er geen tijdswinst is.

Selenium IDE bleek te voldoen aan criteria 1 en 2: de tool is gratis en de testers bleken in staat om zelfstandig ermee te kunnen werken. Criteria 3 bleek echter een issue te zijn; de volgende beperkingen kwamen namelijk aan het licht:

- Grote mate van afhankelijkheid van de GUI
- Testdata zit ingesloten in het testscript
- Herbruikbaarheid van testscripts bleek nauwelijks mogelijk (bijv. bij het testen van verschillende combinaties in het loginvenster)

Daarom heb ik geadviseerd om door middel van een toolselectie naar alternatieven voor Selenium te zoeken vanwege deze beperkingen.

3.3.2 Unittesten en continuous integration

Verder heb ik geadviseerd om het unittesten verplicht te maken, in ieder geval bij Java. Men is daar al bekend met unit testing, echter wordt het te weinig toegepast en is er geen inzicht is in de dekkingsgraad. Een bepaalde dekkingsgraad moet dus verplicht worden en men moet de manier van werken aanpassen zodat er ook tests geschreven worden. Een methode die hierbij kan helpen is *test-driven development*.

Verder kan *continuous integration* ook helpen. Door middel van tools zoals Jenkins kan er centraal worden gebuild en getest en worden de resultaten overzichtelijk weergegeven vanuit een webomgeving. Verder is het met Jenkins ook mogelijk om deployment te automatiseren. Op dat moment werd namelijk de deployment naar de testomgeving bij elke sprint nog handmatig uitgevoerd.

→ Meer informatie over *test-driven development* en *continuous integration* is te vinden in paragraaf 3.4.

Een laatste argument voor *continuous integration* is dat het UTS-team al werkt met Jenkins en hier veel gemak van heeft. Men heeft daar ook aangeraden om het ook bij andere projecten te gebruiken.

Wat de Oracle-afdeling betreft heb ik alleen unittesten geadviseerd, omdat dit daar nog niet wordt toegepast en *continuous integration* teveel veranderingen tegelijk zou betekenen. Bovendien lijkt het daar niet nodig te zijn vanwege de kleine teams en het watervalkarakter (zie paragraaf 3.1.3).

3.4 Literatuurstudie

In de daarop volgende weken heb ik, naast het uitvoeren van de toolselectie (zie paragraaf 3.5), ook een literatuurstudie gedaan. Deze studie was een verdieping in de theoretische achtergrond van het vakgebied van test automation, zoals keyword based testing, data driven testing en het combineren van tools met Agile ontwikkelmethoden. Ook heb ik *best practices* zoals continuous integration & deployment, test-driven development en behaviour-driven development onderzocht en gekeken in hoeverre dit ingezet kan worden binnen Qualogy.

3.4.1 Hoofdvraag en deelvragen

De hoofdvraag van het literatuuronderzoek is:

- Welke methodes en best practices zijn er bekend in de literatuur en kunnen gebruikt worden om het testproces bij Qualogy te automatiseren en meer inzicht geven in de kwaliteit van de software?

De deelvragen zijn:

1. Welke vormen van test automation zijn er?
2. Welke methodes kunnen bijdragen aan beter inzicht in de kwaliteit van de software?
3. Welke methodes kunnen het schrijven van testen stimuleren en integreren in het ontwikkelproces?

3.4.1 Resultaten

De resultaten bestonden uit relevante informatie over de genoemde methodes, waarvan hieronder een korte beschrijving wordt gegeven.

Vormen van test automation: GUI-, API- en code-driven testing

Bij GUI-testen vindt er interactie plaats met de grafische interface van een programma. Dit kan een desktopapplicatie zijn, maar ook een webapplicatie of mobiele applicatie. De meest gebruikte methode is een *record-and-playback tool*, omdat hiervoor geen programmeerkennis vereist is. Het belangrijkste nadeel is dat het afhankelijk is van de GUI en daardoor veel onderhoud kan vereisen bij veranderingen. Zie ook paragraaf 3.5.



Figuur 3.4 - Record-and-playback is een vorm van GUI-testen.

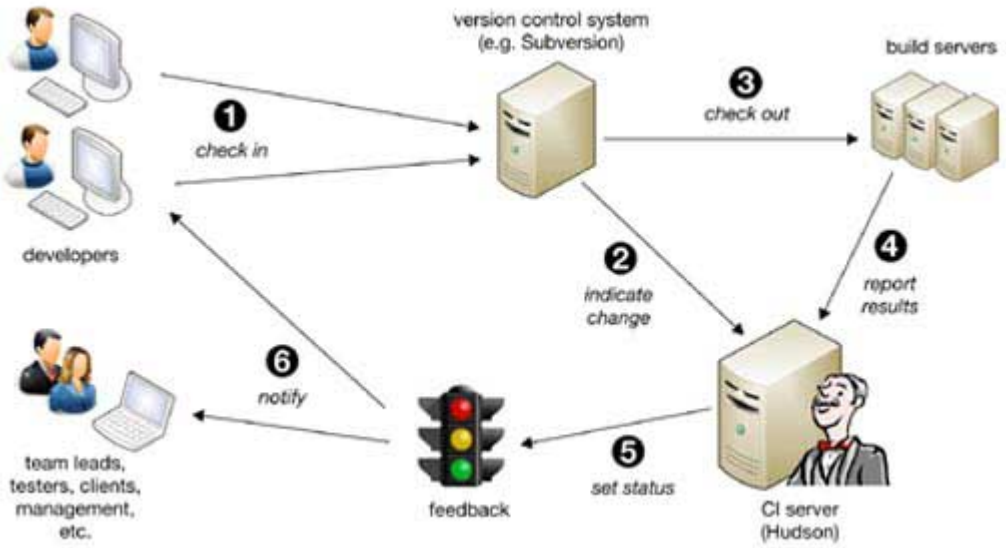
Een andere vorm van GUI-testen is *model-based-testen*. Hierbij wordt er een 'slim' model ontworpen, waarmee automatisch testgevallen ontworpen kunnen worden. Dit vereist echter wel bepaalde expertise en ook hiervoor zijn testscripts nodig om bepaalde gedeelten van het model te implementeren. Voor de situatie bij Qualogy bleek dit niet geschikt, temeer omdat er weinig gemodelleerd wordt bij het ontwerpen van de applicaties en het teveel nieuwe expertise vereist.

Naast GUI-testen zijn er ook API-testen. Deze lijken sterk op GUI-testen, maar hierbij vindt de interactie niet plaats via een GUI maar via een API. In het geval van webapplicaties vinden er bijvoorbeeld HTTP-aanroepen plaats en worden de resultaten gecontroleerd. Een voorbeeld van een tool die dit kan, is BadBoy (zie ook bijlage E: *Toolselectierapport*).

Code-driven testing zijn tenslotte testmethodes waarvan de implementatie wordt verzorgd door een programmeur, eventueel gebruik makend van een API van automation tools (bijv. Selenium WebDriver API). Ook unittesting is een voorbeeld. Deze methode wordt gebruikt om low-level functionaliteiten te testen.

Continuous integration

Continuous Integration (CI) is een methode waarmee de code van meerdere ontwikkelaars geregeld wordt samengevoegd en geïntegreerd naar één omgeving. Het is niet alleen een manier om integratieproblemen te voorkomen, maar wordt ook gebruikt om continue kwaliteitsbewaking toe te passen. De kwaliteit van de code kan namelijk met continuous integration voor iedereen zichtbaar gemaakt worden vanuit een webomgeving. De workflow van *continuous integration* is te zien in figuur 1.3.



Figuur 3.5. Workflow van continuous integration (bron: http://www.methodsandtools.com/)

Continuous integration bestaat uit een aantal *best practices*:

- **Versiebeheer**
  Voor het integreren van de code van de ontwikkelaars, kan een versiebeheersysteem (VCS) worden ingezet. Bekende voorbeelden hiervan zijn Subversion (SVN), Git en Mercurial. De belangrijkste voordelen van een versiebeheersysteem is dat elke wijziging opgeslagen en ongedaan gemaakt kan worden, de code centraal wordt opgeslagen en er binnen een team gelijktijdig aan meerdere versies gewerkt kan worden. Tevens maakt versiebeheer het ook eenvoudiger om OTAP-omgevingen (ontwikkel, test, acceptatie, productie) te beheren. Belangrijk is dat ontwikkelaars dagelijks hun wijzigingen opslaan (*committen*) om conflicten te voorkomen.
- **Automated building**
  Een systeem moet meestal ook eerst nog 'gebuild' worden, voordat het in staat is om succesvol op te starten en te draaien. Dit build-proces bevat bijvoorbeeld het compileren van de broncode, het ophalen van afhankelijkheden en het genereren van statistieken. Het automatiseren van dit proces is meestal onmisbaar bij het toepassen van *continuous integration*, omdat het mislukken van het builden aangeeft dat er nog fouten in de software zit of dat een wijziging niet goed is geïntegreerd. Voor het builden van software zijn er diverse tools, zoals make, Gradle, ant, maven, SCons en Phing.
- **Automated unittesting**
  Uiteindelijk zijn het met name de unittests die inzicht geven in de kwaliteit van de geïntegreerde versie. Daarom worden deze voorafgaand of tijdens het build-proces uitgevoerd. Wanneer de unittests niet slagen, is de software nog niet geschikt voor oplevering aan de accepterende partij (testafdeling of klant). In het kader van kwaliteitsbewaking is het belangrijk dat ook de dekkingsgraad gemeten en inzichtelijk gemaakt wordt.

De voor- en nadelen van continuous integration staan hieronder weergegeven:

Voordelen	**Nadelen**
Snelle feedback naar de ontwikkelaars toe over de kwaliteit, functionaliteit en de systeembrede impact van de code die ze ontwikkeld hebben.	Het kost tijd om het de eerste keer op te zetten.
Constante beschikbaarheid van een stabiele build voor test, demo of release doeleinden.	Er moeten voldoende unittests zijn om de voordelen te kunnen benutten.
	De hardware voor de build machines brengen extra kosten met zich mee

<u>Browser automation / web testing</u>
Bij browser automation kan er een browser worden aangestuurd via een API. Dit kan een werkelijke browser of een virtuele (*headless*) browser zijn. In combinatie met testscripts kan hiermee een end-to-end test worden uitgevoerd zodat er een garantie bestaat dat een functionaliteit daadwerkelijk werkt.

Er is onderzoek gedaan naar de voor- en nadelen van beide type browsers, technieken zoals Selenium Grid en naar testen in de cloud door middel van SauceLabs.

TDD & BDD

Test-driven development (TDD) is een methode waarbij unittesten leidend zijn in het ontwikkelproces. Voordat een functionaliteit ontwikkeld wordt, moet eerst de bijbehorende tests geschreven worden. Deze zal eerst mislukken. Pas wanneer de functionaliteit correct is geïmplementeerd, zal de test slagen. Ook bij gerapporteerde issues moet er eerst een test geschreven worden om de issue te kunnen reproduceren en om later te kunnen bewijzen dat de issue is opgelost.

Enkele argumenten voor test-driven development, zijn:

- Het zorgt dat er meer focus komt op de requirements;
- Het is eenvoudiger om eerst de test te schrijven in plaats van achteraf;
- Het garandeert 100% dekking, wanneer je het goed toepast;
- Achteraf is er vaak geen tijd voor unittesten;
- Het levert testbare code op met duidelijke interfaces;
- Het zorgt voor meer bruikbare code en minder onbruikbare code.
- Bij een issue is het een handige manier om een bug te kunnen reproduceren, bij het oplossen van het issue wordt de test tevens opgenomen in de rest van de test suite.

Een variant hierop is behaviour-driven development (BDD). Deze lijkt erg op TDD, maar daarbij spelen acceptatietesten een meer prominente rol.

→ Meer informatie over alle genoemde methodes is de vinden in hoofdstuk 4 van het adviesrapport.

3.4.2 Conclusie

Door middel van de literatuurstudie ben ik heel veel te weten gekomen over de verschillende soorten testmethodes en tools. Test-driven development en continuous integration zijn interessant om op de Java-afdeling van Qualogy Suriname in te zetten; op de testafdeling kunnen de GUI-testen die op dit moment handmatig uitgevoerd worden deels geautomatiseerd worden door record- en playback tools.

3.5 Toolselectie

In afwachting op de goedkeuring van het plan van aanpak, ben ik in week 2 alvast aan de slag gegaan met het downloaden, installeren en uitproberen van een aantal zelfgekozen tools. Hierbij liep ik in feite op de planning vooruit, maar aangezien dit achteraf een tijdrovend karwei bleek te zijn, was het niet verkeerd om hier tijdig mee te beginnen.

De toolselectie heeft uiteindelijk veel tijd in beslag genomen. Dit heeft te maken met nieuwe inzichten die pas relatief laat ontstonden, waardoor er in feite een nieuwe (definitieve) toolselectie heeft plaatsgevonden. Hieronder worden beide aanpakken besproken.

3.5.1 Eerste aanpak: record & playback

In eerste instantie werd gezocht naar een tool die ingezet kon worden bij de testers op de testafdeling. Dit had als gevolg dat de tool niet teveel technische kennis en programmeervaardigheden mocht vereisen. Het enige type testsoftware dat in een dergelijk geval overblijft, zijn zogenaamde *record and playback* tools. Hiermee kunnen de testers via hun browser een testscript opnemen en later opnieuw afspelen. Een testscript kan één of meer testgevallen bevatten en voert hierbij de nodige checks uit via de GUI van de webapplicatie.

Randvoorwaarden

De randvoorwaarden voor de longlist waren als volgt:

1. Het moet in ieder geval kunnen draaien op Windows 7 en Windows 8
2. De tool moet gratis zijn of weinig kosten (0 - 500 USD eenmalig is acceptabel)
3. Er moeten geen programmeervaardigheden worden vereist van de gebruiker
4. De testscripts moeten flexibel zijn en hergebruik mogelijk maken (data-driven testen)
5. Geschikt binnen een scrum-omgeving, waar het systeem aan verandering onderhevig is

De eerste twee randvoorwaarden zijn door de opdrachtgever bepaald, de overige eisen heb ik zelf vastgesteld op basis van mijn inzicht in de bestaande situatie. Er is vanuit de opdrachtgever nauwelijks input gegeven voor de randvoorwaarden, ik heb hierin dus heel veel vrijheid gekregen.

Bevindingen

Zoals reeds hierboven beschreven, bestond de longlist uit *record-and-playback* tools. In totaal waren er 12 tools, welke in ongeveer 3 weken tijd zijn getest in een virtuele omgeving. Hiervoor is er gebruik gemaakt van evaluatielicenties (*trials*) en is er geïnventariseerd wat de mogelijkheden waren en in hoeverre de gevonden tools voldeden aan de randvoorwaarden.

De tools uit de originele longlists is te vinden in tabel 3.1.

	Uitgever	Platforms	Kosten licentie
Selenium	-	Win/Mac/Linux	gratis
Sahi	-		gratis $495 eenmalig (pro)
Badboy	Badboy Software	Windows	$45 eenmalig
TestComplete	Smartbear Software	Windows	$1.000 eenmalig
Unified Functional Testing	Hewlett Packard	Windows	onbekend
Robot Framework	robotframework.org	Win/Mac/Linux	open source
Twist	Thoughtworks Inc.	Win/Mac/Linux	$99 / jaar
iMacros	Ipswitch Inc.	Windows	$495 eenmalig
QF-Test	QFS GmbH	Win/Mac/Linux	$2.584 eenmalig
Ranorex Studio	Ranorex GmbH	Windows	$890 eenmalig
Test Studio	Telerik	Windows	$169 / maand
SoapUI **(Pro)**	Smartbear Software	Win/Mac/Linux	gratis (community) $399-449 / jaar (pro)
Silk Test	Borland Software (MicroFocus)	Windows	$4.500 - 9.100
Tricentis Tosca Testsuite	Tricentis GmbH	Windows	onbekend
Rational Functional Tester	IBM	Windows/Linux	Vanaf $3.030 eenm.
TestDrive	Original Software		onbekend
SOAtest	Parasoft		onbekend

Tabel 3.1 - Originele longlist voor het selecteren van een record & playback tool.

Na enkele weken met diverse tools te hebben gewerkt bleken er structureel een aantal problemen te zijn met de gevonden tools:

- **Afhankelijkheid van GUI**
  Het belangrijkste nadeel van record and playback is dat er een grote mate van afhankelijkheid is met de GUI. Zeker in een Scrum omgeving is dit lastig, omdat hierbij het systeem vaak aan verandering onderhevig is. Dit betekent dat de testscripts vaak aangepast moeten worden, wat veel onderhoud kan opleveren.
- **Testdata zit ingesloten in het testscript**
  Ook zou het goed zijn als de testdata gescheiden kan worden van het testscript, zodat een scenario verschillende keren doorlopen kan worden met verschillende testdata als input. Dit bleek bij de meeste tools niet te kunnen.

- **Herbruikbaarheid van testscripts**
  Aangezien er vanwege de hoeveelheid testgevallen veel herhaling zit in de testscripts, met name bij grenswaardenanalyses en beslistabeltesten, ontstond de behoefte om stukken testscripts te kunnen her- gebruiken. Met name de eenvoudige *record and playback* tools bleken hier niet toe in staat, en bij de wat uitgebreidere (betaalde) software bleek dit vaak slechts beperkt te werken of ging dit heel omslachtig.
- **Kwaliteit van HTML**
  Bij het KvK-project speelde er nog een ander aspect, namelijk de opbouw en kwaliteit van de HTML-code van de webapplicatie. De frontoffice van dit systeem is gebouwd met Qafe en bevat veel AJAX en JavaScript in combinatie met verborgen input-velden van formulieren, ontbrekende formuliertags, etcetera. Dit zorgde ervoor dat er in Selenium IDE veel met 'waits' gewerkt moest worden en bij BadBoy zorgde het zelfs ervoor dat het recorden soms niet goed ging.
- **Dure en 'zware' tools**
  Een ander belangrijk probleem is dat testsoftware die voor genoemde problemen enigszins oplossingen kan aanbieden, vaak erg duur bleek te zijn. En dat terwijl één van de eisen was dat de software bij voorkeur gratis, dan wel goedkoop moest zijn. Uiteindelijk kwam ik uit bij uitgebreide enterprise softwarepakketten van Borland, IBM of SmartBear, welke naast het fikse prijskaartje ook nog eens behoorlijk traag kunnen zijn bij het recorden en afspelen (terwijl deze functie het belangrijkst is).

<u>Discussie en vervolg toolselectie</u>
Deze bevindingen brachten de volgende fundamentele vragen naar boven:

- In hoeverre is het überhaupt mogelijk om een test tool te vinden waarbij niet geprogrammeerd hoeft te worden? Is het niet beter om ontwikkelaars in te zetten voor het maken van de testscripts?
- Wat is beter: veel tijd investeren in duurzame en flexibele testscripts door middel van uitgebreide software, of juist met eenvoudige tools snelle testscripts te maken zodat het niet erg is wanneer een testscript aangepast hoeft te worden?

Deze vragen heb ik in eerste instantie voorgelegd aan mijn bedrijfsmentor, maar aangezien deze niet veel ervaring heeft met test automation, heb ik tenslotte overleg gehad met een collega uit Nederland. Deze heeft mij veel advies en tips gegeven en gewezen op een aantal testtools waarmee men in Nederland goede ervaringen had.

Dankzij deze nieuwe inzichten heb ik besloten om de toolselectie opnieuw uit te voeren, maar met andere randvoorwaarden en een heel ander uitgangspunt. Het idee was om naar een constructie te zoeken waarbij testers en ontwikkelaars samenwerken bij het maken van de testscripts. De testers zouden dan door middel van eenvoudige *record and playback* tools zoals Selenium IDE snel een prototype van een testscript kunnen maken, welke vervolgens door de ontwikkelaars verbeterd, geoptimaliseerd en afgewerkt kunnen worden. Bovendien is er dan een integratie van de webtesten in de unittestomgeving mogelijk, zodat ook de webtesten een onderdeel kunnen uitmaken van *continuous integration*.

3.5.2 Nieuwe aanpak: framework van tools

Met het nieuwe gezichtspunt is er opnieuw een toolselectie uitgevoerd, waarbij er een framework met meerdere tools is voorgesteld dat binnen Qualogy gebruikt kan worden.

Werkwijze

Bij het uitvoeren van de toolselectie worden de volgende stappen uitgevoerd.



1. **Longlist**
   In deze fase wordt er op basis van algemene randvoorwaarden een lijst opgesteld met alle tools die in aanmerking komen voor de toolselectie.
2. **Shortlist**
   In deze fase worden er gedetailleerde selectiecriteria opgesteld en op basis daarvan wordt de lijst met tools d.m.v. scores beperkt tot een shortlist.

Oorspronkelijk was er ook een **proof of concept** en een **pilot** gepland, om de tools daadwerkelijk uit te kunnen testen. Om tijd te besparen wordt dit echter niet meer uitgevoerd. Het advies zal bestaan uit een aantal tools uit de shortlist en er zal tijdens het implementeren definitieve keuzes gemaakt worden aan de hand van evaluaties van gebruikers.

Indeling tools

De grootste uitdaging was om uit de zeer grote longlist aan vele diverse tools een kloppende indeling te maken. Immers: sommige tools bleken alternatieven van elkaar te zijn, terwijl andere tools elkaar juist weer aanvulden. De volgende categorieën zijn er uiteindelijk vastgelegd:

- **Web unittesting**: ook wel *functional testing* of *acceptance testing* genoemd. Dit zijn end-to-end testen die vanuit een unittestomgeving een of meer browsers kan aansturen.
- **Browser automation**: hierbij wordt gekeken naar het aansturen van headless browsers en echte browsers via een API, eventueel met een extra abstractielaag.
- **BDD tools:** tools waarmee specificaties (requirements) geïntegreerd kunnen worden in het testproces, door middel van de BDD (behaviour driven development) methodiek.
- **Record-and-playback**: er is gekeken of er geschikte *record-and-playback* tools zijn waarmee testers kunnen prototypen en code kan genereren die vervolgens in de unittestomgeving geïntegreerd kan worden.
- **Test frameworks**: ook werd er gezocht naar een eventueel bestaand framework dat meerdere tools met elkaar kan integreren en extra functionaliteit aanbiedt zoals testplanning en rapportage.

Randvoorwaarden

Voor het opstellen van de longlist gelden er per categorie de volgende randvoorwaarden. De randvoorwaarden zijn gebaseerd op de bestaande situatie, zoals het kennisniveau van de ontwikkelaars en de reeds gebruikte software.

- Unit testing
  - Het framework moet integreren in de bestaande IDE's
  - Op dit moment wordt JUnit al gebruikt, dus wordt er gekeken naar een xUnit framework waarbij er ook web tests uitgevoerd kunnen worden (end-to-end), in combinatie met een browser automation driver.
- BDD tools
  - De tool moet integreren in de bestaande IDE's
  - De specificaties moeten in plain text geschreven en uitgelezen kunnen worden.
  - Het moet mogelijk zijn om code 'stubs' te genereren.
- Browser automation
  - De driver moet kunnen werken met recente versies van Internet Explorer, Mozilla Firefox en Google Chrome. Opera en Safari zijn niet verplicht.
  - In-browser emulators zou mooi zijn, maar in ieder geval headless browsers moeten ondersteund worden.
  - Headless browsers moeten ondersteuning bieden voor WebDriver, voor een eventuele Selenium Grid-configuratie.
- Test frameworks
  - Integratie met andere tools is het belangrijkst.
  - Er moet een webomgeving beschikbaar zijn, zodat het toegankelijk is voor het gehele team.
- Record & playback
  - Moet compatibel zijn met Windows, bij voorkeur ook op Linux / Mac OS.
  - Moet tenminste één van de volgende browsers ondersteunen: IE, Firefox of Chrome.
  - Er mogen geen programmeervaardigheden vereist worden.
  - Rapportage moet mogelijk zijn, eventueel met screenshots.
  - Bij voorkeur moeten testscripts hergebruikt kunnen worden (modulaire opbouw).

Resultaten

De uiteindelijke shortlist wordt hieronder weergegeven. Voor een uitgebreide bespreking van de tools, zie *Bijlage E - Toolselectierapport*.

Web unit testing	**BDD tools**	**Browser automation**	**Test frameworks**	**Record & playback** & enterprise software
Optie 1 JWebUnit *Optie 2* Canoo WebTest Geb	*Optie 1* Spock *Optie 2* JBehave	*In-browser emulators* Selenium (WebDriver) *Headless browsers* HtmlUnit PhantomJS	Thucydides	Badboy Selenium IDE Oracle Application Testing Suite

3.6 Adviesrapport

Op basis van de eerder uitgevoerde inventarisatie, literatuurstudie en toolselectie is er van week 8 t/m 10 een adviesrapport opgesteld.

3.6.1 Overzicht

De volgende onderdelen zijn opgenomen in het adviesrapport:

- **Theorie**: hierin wordt uitleg gegeven over de methodes en best practices die ter sprake komen;
- **Onderzoek**: de resultaten van de interviews, literatuurstudie en de toolselectie worden besproken;
- **Advies**: op basis van alle bevindingen en de conclusie van de toolselectie wordt er een advies aangeboden voor de door te voeren wijzigingen en de te implementeren tools;
- **Implementatie**: er wordt aangegeven op welke wijze het advies geïmplementeerd kan worden.

3.6.2 Feedback

Het adviesrapport gericht is primair aan het management van Qualogy, welke niet van alle best practices en methodes op de hoogte is. Daarom heb ik besloten te beginnen met een theoretisch hoofdstuk. Van mijn manager heb ik echter na oplevering begrepen dat hij dit wel "een vrij saai hoofdstuk" vond. Tijdens een latere presentatie had ik echter wel door dat deze uitleg niet overbodig was: niet iedereen wist namelijk wat continuous integration was. Verder was men bij het management ook niet altijd even goed op de hoogte van de bestaande situatie, zoals dat men deployment nog handmatig doet en dat er nauwelijks unittesten worden gebruikt. Ook werd de noodzaak duidelijk dat de Java-teams onderling meer kennis moeten uitwisselen en dat er vanuit het management meer tijd gereserveerd moet worden voor unit tests.

Op het adviesrapport is relatief weinig feedback gekomen. Er was vanuit het management vooral behoefte aan resultaten. Mijn bedrijfsmentor gaf aan dat het allemaal nog erg theoretisch was en er nog niets concreets was opgeleverd. Daarom hebben we besloten om met de implementatie te beginnen en pas bij het eindadvies te kijken naar eventuele geboekte resultaten.

3.6.4 Uitloop planning

Het opleveren van het adviesrapport heeft ongeveer 3 weken uitloop gehad. Dit heeft vooral te maken gehad met nieuwe inzichten tijdens het uitvoeren van de toolselectie, waardoor deze langer duurde. Vanwege de uitloop is de implementatie aangepast en vooral beperkt tot het KvK-project. De nieuwe planning voor het vervolgtraject is opgenomen in het adviesrapport en ziet er als volgt uit:

Periode / datum	Activiteit(en)	Opleveren
week 11 27 okt - 31 okt	Grails omgeving inrichten Inwerken bij KvK	
week 12 3 okt - 7 nov	Jenkins opzetten (unit testing / deployment / code inspection)	- Werkende Jenkins voor KvK - Documentatie van de configuratie
week 13 10 okt - 14 nov	Unit-testing invoeren bij Java / Oracle	- Presentaties unit testing - Document met richtlijnen / eisen
week 14 - 15	Web testing opzetten (Selenium)	- Werkende Selenium tests in KvK

17 okt - 28 nov		- Selenium Grid (optioneel)
week 16 1 - 5 dec	Implementatierapport schrijven	- Implementatierapport
week 17 8 - 12 dec	Aan stageverslag werken Eventueel: acceptance testing invoeren	
nader te bepalen	Evaluatie met opdrachtgever	
week 18 15 dec - 19 dec	Afronden project, documentatie en stageverslag	- Volledig afstudeerdossier (rapportage en stageverslag)

4. Implementatiefase

In dit hoofdstuk komt de implementatie van de in het adviesrapport beschreven tools en methodes aan bod.

Vanwege de uitloop in de planning hebben we besloten om de implementatie in eerste instantie alleen bij het KvK-project uit te voeren. Met ingang van de implementatiefase ben ik dan ook verhuisd naar de kamer waar de ontwikkelaars van dit project werkzaam zijn. Om het later ook op andere afdelingen toe te kunnen passen wordt er een implementatierapport geschreven waarin duidelijk vermeld staat hoe de tools geconfigureerd zijn.

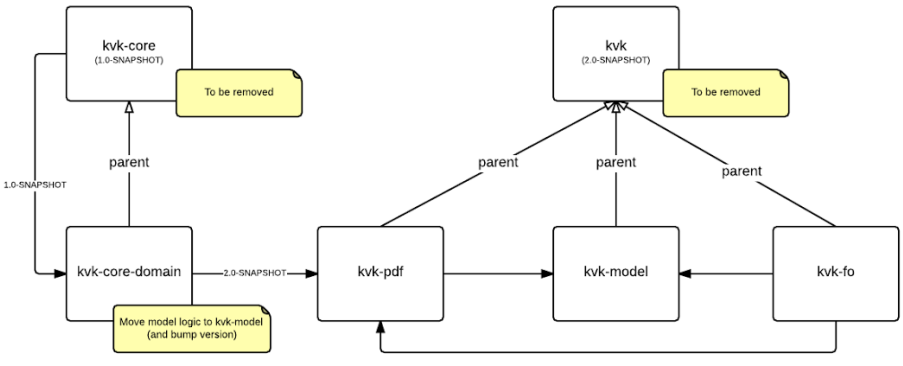
4.1 Voorbereiding

Het KvK-project is een project dat al wat langer loopt: in oktober 2013 is het huidige team gestart met de frontoffice. Deze werd gebouwd in Qafé, een platform/framework hoofdzakelijk door Qualogy zelf ontwikkeld. Later is er een projectmanager uit Nederland geweest die besloten heeft om voor de backoffice een ander framework te gebruiken; dit werd uiteindelijk Grails. Met de backoffice is men in het voorjaar van 2014 gestart.

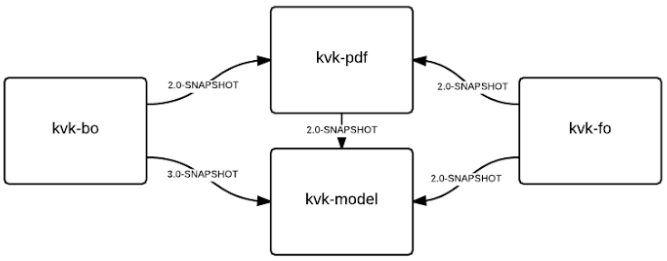
Doordat de ontwikkelaars bij KvK nog relatief weinig ervaring hebben met software development (1 - 3 jaar), is het kennisniveau nog wel wat aan de lage kant. Dit was terug te zien in de manier waarop het project geordend was en de kwaliteit van de code.

De eerste dagen bij het KvK-project heb ik dan vooral besteed aan het refactoren van de code, het opnieuw organiseren van de packages, en het opschonen van configuratiebestanden. Ook heb ik mijn ontwikkelomgeving ingericht en heb ik mij verdiept in de werking van Maven, Grails en andere ontwikkeltools. Dit paste gelukkig wel in de planning, ik had hier namelijk een week voor gereserveerd.

Zoals te zien is in figuur 4.1 waren de packages op een wat omslachtige manier ingericht. Deze heb ik opnieuw georderd, zoals in figuur 4.2 te zien is.



Figuur 4.1 KvK Package Diagram (oude situatie)



Figuur 4.2 KvK Package Diagram (nieuwe situatie)

Met deze nieuwe structuur zijn het aantal dependencies verlaagd en kunnen alle packages onafhankelijk gebuild worden. Het belangrijkste voordeel is dat de mappenstructuur van SVN overzichtelijker is en de projecten los van elkaar in Jenkins geconfigureerd kunnen worden.

4.1.1 Maven

Verder heb ik leren werken met Maven. In eerste instantie dacht ik dat Maven voornamelijk een package manager was, zoals NuGet uit Microsoft .NET en Composer in PHP. Dat is zo, maar Maven is meer dan dat. Met Maven kun je het hele proces van *builden* tot in detail configureren, inclusief eventuele aanvullende acties zoals unit testing, code coverage, rapportage en deployment.

Maven gebruikt een "Project Object Model" (POM) om alles te configureren. Dit is een XML-bestand in de root van je project. In de POM staan verder de afhankelijkheden met andere modules en componenten, waaruit de volgorde van bouwen bepaald wordt. In onderstaand figuur staat er een voorbeeld van een *pom.xml* bestand.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <!-- The Basics -->
  <groupId>sr.qualogy.kvk</groupId>
  <artifactId>kvk-pdf</artifactId>
  <version>2.0-SNAPSHOT</version>

  <properties>...</properties>
  <dependencies>...</dependencies>
  <modules>...</modules>

  <!-- Build Settings -->
  <build>...</build>
  <reporting>...</reporting>

  <!-- More Project Information -->
  <name>...</name>
  <description>...</description>

  <!-- Environment Settings -->
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <profiles>...</profiles>
</project>
```

4.1.2 Groovy & Grails

Ook heb ik met Groovy en Grails leren werken. Groovy is een programmeertaal voor het Java-platform en lijkt op Java maar heeft een eenvoudigere syntax en ondersteunt een aantal extra zaken, zoals static type checking en een eenvoudigere syntax:

```
["Rod", "Carlos", "Chris"].findAll{it.size() <= 4}.each{println it}
```

Ook kun je getters en setters achterwege laten, deze worden impliciet gegenereerd (properties). Verder is er een prettige syntax voor closures:

```
def list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
def odds = list.findAll { i -> i % 2 }
```

Grails is een Groovy-framework voor webapplicaties. Het is gebouwd op Java Spring en maakt veelvuldig gebruik van het principe 'convention over configuration'. Dit houdt in dat er weinig geconfigureerd hoeft te worden zolang je volgens bepaalde conventies volgt.

Een controller kan bijvoorbeeld zou eenvoudig zijn als:

```
package org.bookstore

class HelloController {
    def world() {
        render "Hello World!"
    }
}
```

Hieronder een wat uitgebreider voorbeeld van een controller voor het weergeven van boeken. De controller heeft twee *actions*, namelijk een lijstweergave en een detailweergave. Bij beide actions wordt er automatisch een template gerenderd, die volgens de conventie in een bepaalde directory moeten staan.

```
package org.bookstore

class BookController {

    def list() {
        [books: Book.list(), authors: Author.list()]
    }

    def show() {
        [book: Book.get(params.id)]
    }
}
```

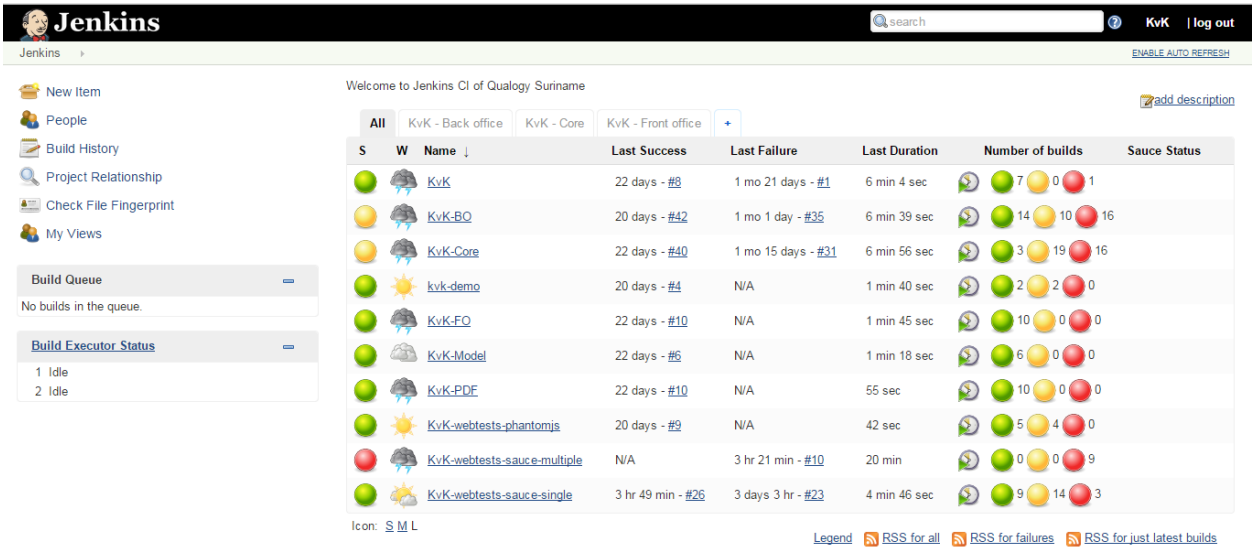
4.2 Continuous integration

Voor het inrichten van continuous integration heb ik besloten om Jenkins te gebruiken, omdat deze ook bij het andere Java-team werd gebruikt. Deze heb ik eerst lokaal geïnstalleerd en geconfigureerd, omdat er door systeembeheer nog geen server was ingericht. Voor het inzien van code coverage heb ik Cobertura gebruikt, waarvoor zowel een Maven-plugin als een Jenkins-plugin voor beschikbaar was.

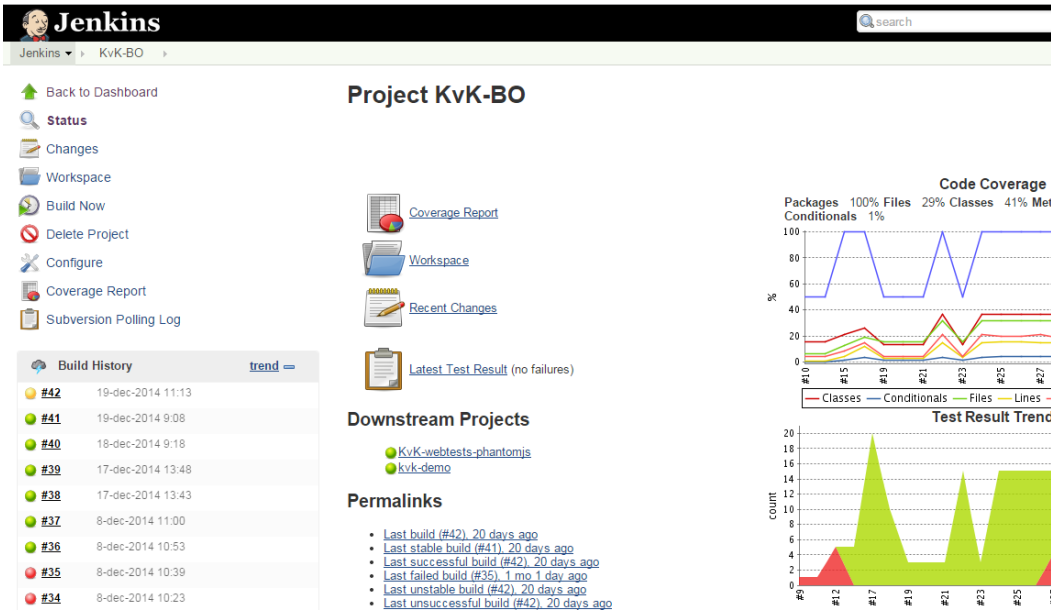
Verder was er een manier nodig waarop de packages van de projecten intern opgeslagen konden worden in een repository. Hiervoor heb ik in eerste instantie Artifactory gebruikt. Nadat ik echter iemand van het UTS-project had gesproken, waar ze al met Jenkins werken, bleek dat zij al met Nexus werken. Om consistent te blijven met het andere team is er daarom besloten om Artifactory te vervangen door Nexus.

Het duurde vrij lang voordat systeembeheer een server had ingericht, dit heeft ruim twee weken geduurd. Daarom heb ik alles eerst lokaal opgezet. Het inrichten van Jenkins is redelijk snel uitgevoerd, binnen 2 dagen draaide het. Nadat systeembeheer de server had klaargezet, heb ik Jenkins daar ook opgezet. Ik kon de configuratie overnemen van mijn lokale installatie. Dit heeft wel voor wat vertraging gezorgd.

In week 15 is er een demo gegeven van Jenkins. Hierbij waren zowel ontwikkelaars, als testers en mijn manager aanwezig. Tijdens de demo is uitleg gegeven over het nut van unit testen, hoe Jenkins inzicht biedt in de kwaliteit en hoe de ontwikkelaars een deployment kunnen uitvoeren.



Figuur 4.3 Het startscherm van Jenkins, met alle "jobs".



Figuur 4.4 Het overzichtsscherm van een project binnen Jenkins

4.3 Unit testing

Zoals in paragraaf 4.1 reeds werd genoemd, bestaat het KvK-project uit meerdere packages:

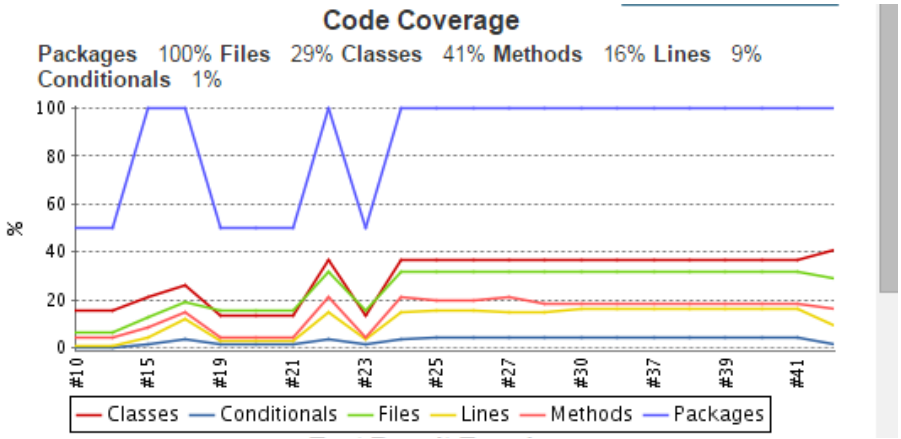
- kvk-fo: de frontoffice, gebouwd in Java i.c.m. Qafé;
- kvk-pdf: een Java-library voor het genereren van pdf's;
- kvk-model: de business layer voor de frontoffice met validatie en een data-abstractielaag;
- kvk-core: de business layer voor de backoffice m.b.v. Java Spring/Hibernate/JPA;
- kvk-bo: de backoffice, gebouwd in Grails.

Met behulp van Cobertura was het mogelijk om de dekkingsgraad inzichtelijk te maken. Deze was op dat moment als volgt:

	Classes	**Methods**	**Lines**
kvk-fo	25% (2/8)	13% (18/141)	13% (86/673)
kvk-pdf	0% (0/18)	0% (0/233)	0% (0/2675)
kvk-model	80% (39/49)	46% (750/1620)	57% (2022/3529)
kvk-core	60% (140/235)	5% (253/5449)	5% (546/11246)
kvk-bo	0% (0/14)	0% (0/64)	0% (0/349)

Aangezien de frontoffice zeer binnenkort gereed komt, hebben we besloten om eerst de backoffice van unit- en integratietests te voorzien. Bovendien is de backoffice in Grails gebouwd, welke uitstekende ondersteuning biedt voor tests; in tegenstelling tot de frontoffice, waar unittests voor de Java-classes vrij moeilijk te maken zijn vanwege het gebruik van Qafé.

Na drie dagen van code refactoren, opschonen en unittesten schrijven was de dekkingsgraad van het **kvk-bo** project verhoogd naar 16% (57/354). Deze trend hebben we doorgezet, in figuur 4.5 is te zien wat het resultaat was na 2 weken.



Figuur 4.5 Het verloop van de dekkingsgraad is te zien binnen Jenkins.

Bij het schrijven van de unit tests heb ik gebruik gemaakt van Spock als framework.
Zoals de toolselectie al aangaf, is dit een zeer geschikt om te gebruiken binnen een Groovy-omgeving.
Hieronder staan een aantal voorbeelden van testscripts.

```
void setup() {
    bedrijf1 = new KvkOndernemingDTO(ondernemingId: 1, kvkNrBedrijf: 1234, handelsnaam: 'Lorem');
    bedrijf2 = new KvkOndernemingDTO(ondernemingId: 2, kvkNrBedrijf: 5678, handelsnaam: 'Ipsum');
    bedrijf3 = new KvkOndernemingDTO(ondernemingId: 3, kvkNrBedrijf: 2233, handelsnaam: 'Dolor');

    controller = new BedrijvenController(
            bedrijfFacade: Mock(BedrijfFacade)
    )
}

void testIndex() {
    when:
    def result = controller.index()

    then:
    result.title == 'Bedrijven'
}

void testAll() {
    when:
    controller.all()

    then:
    1 * controller.bedrijfFacade.geefAlleBedrijven() >> [bedrijf1, bedrijf2, bedrijf3]

    and:
    ((JSONArray) controller.response.json).length() == 3;
}

void testSearchKvkNummer() {
    given:
    controller.params.bedrijf = [
        kvknummer: ' 5678 '
    ]

    when:
    controller.search()

    then:
    1 * controller.bedrijfFacade.geefBedrijf('5678') >> [bedrijf2]

    and:
    ((JSONArray) controller.response.json).length() == 1;
}
```

Enkele zaken die opvallen:

- Spock ondersteunt *given-when-then syntax* in de code en is daarom geschikt als tool voor behaviour driven development;
- Eenvoudige methode voor *mocking*: de syntax `1 * someObject.method()` betekent dat er getest wordt of de methode exact één keer wordt aangeroepen.
- Eenvoudige methode voor *stubbing*: de syntax `someObject.method() >> "someValue"` betekent dat de methode van het mockobject een specifieke waarde moet teruggeven.

4.4 Browsertesten (web testing)

Tenslotte wilde ik nog een aantal browsertesten implementeren. Tijdens het opzetten van continuous integration en het schrijven van de unit testing had ik al wat vertraging opgelopen, zodat ik voor web testing nog maar een krappe twee weken had. Dit is daardoor vrij summier opgeleverd; in ieder geval heb ik het werkend gekregen en heb ik hiervan een demonstratie kunnen geven.

4.4.1 Uitdagingen

Ik kreeg te maken met de volgende uitdagingen.

Allereerst wilde ik de browsertesten integreren in het Grails-project van de KvK backoffice. Dit leverde erg veel problemen op, vanwege een Grails-plugin die vol zat met vreemde bugs. Ik heb geprobeerd om Grails te upgraden, wat vervolgens weer andere bijeffecten had. Na een week lang ermee te hebben gestoeid, heb ik alles overboord gegooid en ben ik in een los Java-project opnieuw begonnen. Dat ging stukken beter.

Er zijn prachtige libraries beschikbaar die de syntax van Selenium sterk vereenvoudigen (zie ook het Toolselectierapport). Voorbeelden zijn Helium en Geb. Beide zijn zeer gebruiksvriendelijk in gebruik. Echter zocht ik naar een methode waarbij testers hun testscript met Selenium IDE kunnen opnemen, en deze vervolgens kunnen exporteren naar programmacode. Selenium IDE ondersteunt echter alleen JUnit als testframework voor Java (zie paragraaf 3.2). Voor Helium en Geb heb ik echter geen zogenaamde "Selenium formatter" kunnen vinden. Daarom ben ik met JUnit verder gegaan.

Ik heb eerst geprobeerd om de Selenium testscripts van de KvK frontoffice te te integreren in de JUnit omgeving. Dit leek in eerste instantie goed te gaan, maar vanwege de snelheid van het testscript begon de *system under test* (de applicatie) erg vreemd te doen. Dit kreeg ik niet opgelost. Dit heeft te maken met het feit dat de frontoffice is gebouwd met Qafé. Qafé is een in-house ontwikkeld framework, waarbij de frontend bestaat uit een single-page applicatie die met de backend communiceert door middel van remote procedures calls m.b.v. AJAX-aanroepen. In feite komt het erop neer dat Qafé van zichzelf een aantal beperkingen heeft waardoor de Selenium testscripts niet te snel uitgevoerd kunnen worden. Ik heb dit niet kunnen verhelpen. Uiteindelijk heb ik daarom geadviseerd om Qafé-applicaties nog semi-automatisch te testen vanuit Selenium IDE en de automatische browsertesten alleen voor niet-Qafé applicaties te gebruiken, zoals applicaties die met Grails zijn gebouwd.

4.4.2 Voorbeelden van testscripts

Hieronder staat een voorbeeld van een eenvoudig Selenium testscript in een JUnit-omgeving:

```
/**
 * @author Dirk Luijk <dluijk@qualogy.sr>
 */
public class LoginTest {
    private WebDriver driver;
    private String baseUrl;

    @Before
    public void setUp() throws Exception {
        DesiredCapabilities cap = new DesiredCapabilities();
        cap.setCapability(PhantomJSDriverService.PHANTOMJS_EXECUTABLE_PATH_PROPERTY, "/usr/local/bin/phantomjs");

        driver = new PhantomJSDriver();
        baseUrl = "http://172.19.2.89:8080/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    }

    @Test
    public void testLoginSuccess() throws Exception {
        login("admin", "admin");
        assertEquals("Ingelogd als Admin", driver.findElement(By.cssSelector("td > p")).getText());
        driver.findElement(By.linkText("Uitloggen")).click();
    }
```

Zoals in de *setUp()* methode is te zien, werkt dit testscript met PhantomJS als headless browser. Deze browser bleek namelijk het snelste van alle browsers te zijn:

PhantomJS: ~ 10 sec
HtmlUnit: ~ 20 sec
Mozilla Firefox: ~ 55 sec

Bovendien werkt PhantomJS als headless browser ook vanuit de build-omgeving van Jenkins. Voor het uiteindelijke test-project heb ik het zó gemaakt dat je via een parameter kan meegeven in welke browser je de tests wilt runnen:

Firefox:

```
mvn test -Dbrowser=firefox
```

PhantomJS:

```
mvn test -Dbrowser=phantomjs
```

Zo kan een developer kiezen of hij de tests in een echte browser zoals Firefox wilt uitvoeren, of in een snellere headless browser zoals PhantomJS. De volgende browsers worden ondersteund: Mozilla Firefox, Google Chrome, HtmlUnit, PhantomJS, SauceLabs.

In een abstracte klasse *WebTest* plaatste ik de volgende code in de *setUp()* methode:

```
if (browser.equals("firefox")) {
```

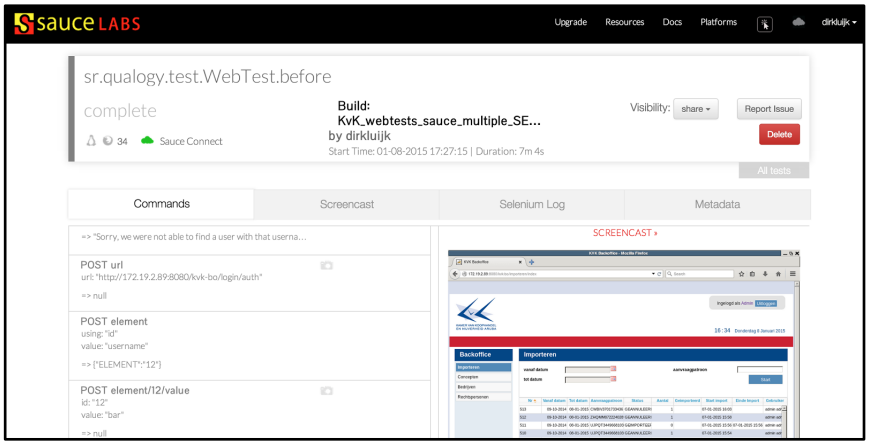
```
    wd = new FirefoxDriver();
} else if (browser.equals("chrome")) {
    wd = new ChromeDriver();
} else if (browser.equals("htmlunit")) {
    wd = new HtmlUnitDriver(true);
} else if (browser.equals("phantomjs")) {
    DesiredCapabilities dcaps = new DesiredCapabilities();
    dcaps.setCapability(
        PhantomJSDriverService.PHANTOMJS_EXECUTABLE_PATH_PROPERTY,
        Config.get("phantomjs.binary.path"));
    wd = new PhantomJSDriver(dcaps);
} else if (browser.equals("saucelabs")) {
    wd = SeleniumFactory.createWebDriver();
}
```

De browsertesten draaiden prima, niet alleen lokaal maar ook in Jenkins.

4.4.3 Selenium Grid

Vervolgens had ik nog een laatste uitdaging: het testen in werkelijke browsers vanuit de Jenkins-omgeving. Omdat de Jenkins-server op CentOS (Linux) draaide, waren er lokaal geen normale browsers zoals Google Chrome of Firefox beschikbaar. De enige oplossing is om een Selenium Grid te bouwen. Met een dergelijke opstelling is het mogelijk om op afstand (bijv. met virtual machines) meerdere *nodes* te laten testen. Deze *nodes* zijn machines waarom verschillende besturingssysteem/browser combinaties draaien. Meer uitleg hierover kunt u vinden in paragraaf 3.2.3 van het toolselectierapport.

Helaas was er geen tijd meer om een eigen Selenium Grid op te zetten. Wel heb ik nog kunnen experimenteren met SauceLabs. Dit is een cloudservice waarbij er eigenlijk een Selenium Grid via de cloud wordt aangeboden. Door middel van een beveiligde "tunnel" kan de cloudservice in je lokale netwerkomgeving de applicatie testen met allerlei verschillende browsers, waaronder ook mobiele browsers.
Aangezien er een gratis trial werd aangeboden van 90 dagen, Ik heb dit vooral uit nieuwsgierigheid nog even op de valreep uitgeprobeerd, maar veel tijd had ik er niet meer voor. Ik heb het werkend gekregen, maar het was wel vrij traag. Dit kan zijn omdat de internetverbinding in Suriname niet zeer snel is, en alles ook nog eens via een beveiligde tunnel liep. Dat is precies ook de reden dat men bij Qualogy Suriname niet alles in de cloud wil hebben draaien.



Figuur 4.6 SauceLabs, testen op verschillende browsers via de cloud

5. Oplevering

Gedurende de laatste 1,5 week van het project heb ik me bezig gehouden met het opleveren van de laatste producten en het afronden van de werkzaamheden. De belangrijkste aandachtspunten waren:

- Duidelijk aangeven wat de vervolgstappen zijn voor de organisatie;
- Voldoende kennis overdragen aan collega's, zodat zij zelfstandig verder kunnen;
- De resultaten presenteren aan het management.

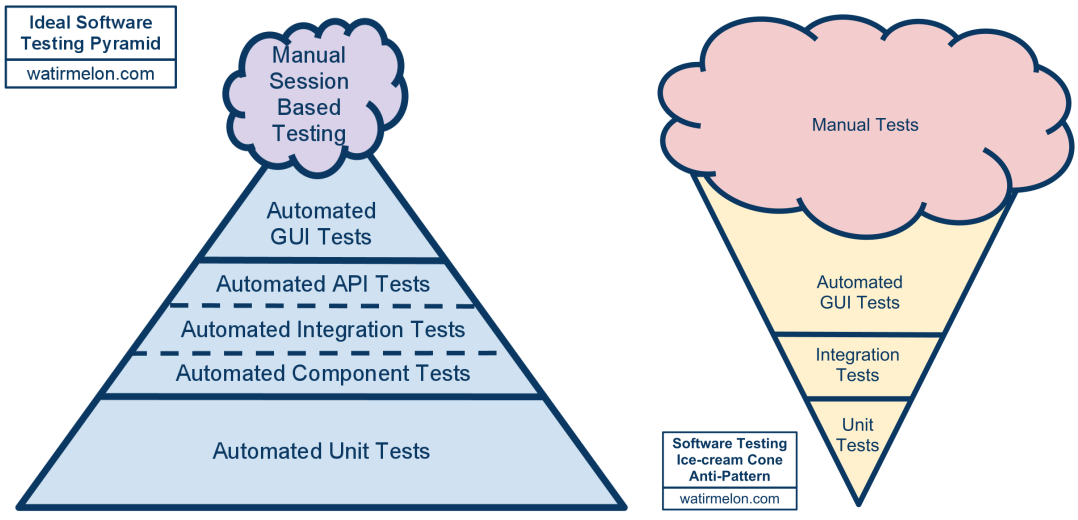
5.1 Overdracht aan collega

Een collega bij het KvK-project, Jozua Herfst, heb ik uitleg gegeven over de werking van Jenkins en hoe er met test tools zoals Selenium gewerkt kan worden. Tevens heb ik een implementatierapport opgeleverd, waarin deze zaken ook staan. Ook heb ik aangegeven hoe men in de toekomst continuous deployment ingevoerd kan worden.

5.2 Presentatie afstudeeropdracht

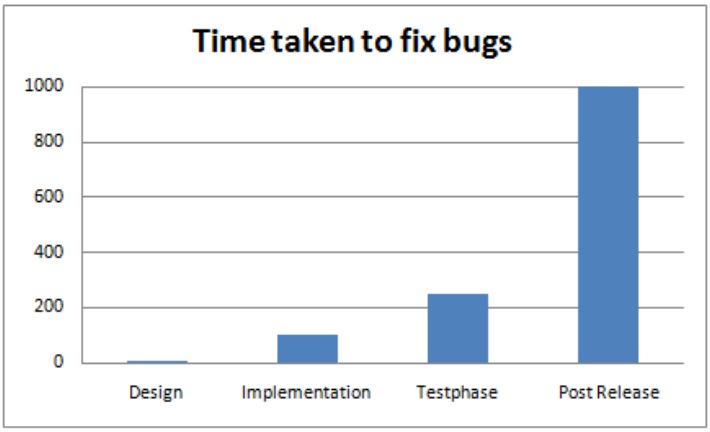
Op de laatste dag heb ik een presentatie gegeven aan het management, waarin ik liet zien welke resultaten waren bereikt en welke actiepunten nog open stonden. Hierbij waren aanwezig: Ray Quartier (mijn mentor), Purcy Sweeb (mijn manager) en Nilesh Bishesar (directeur). Omdat ik uiteindelijk ook verantwoording moest afleggen aan Nilesh en hij niet betrokken is geweest bij het project, heb ik een nogmaals een duidelijke samenvatting gegeven van de opdracht, en kort stilgestaan bij de doelstelling, de verrichte werkzaamheden en de bereikte resultaten.

Het belang van goede unit testen heb ik aangetoond door gebruik te maken van een zogenaamde "testpiramide". Hiermee liet ik zien dat weinig unittesten en veel handmatig testen zorgt voor veel tijdverlies aan het eind van een project. Met geautomatiseerde unittests als basis heb je een solide basis waarmee veel tijd wordt bespaard.



Figuur 5.1 - De ideale testpiramide (links) en de situatie bij Qualogy (rechts).

Ook is het zo dat bugs in de applicatie die pas na oplevering worden ontdekt, veel meer tijd kosten dan bugs die tijdens het ontwikkelen of het testen worden ontdekt.



Figuur 5.2 - Hoe later een fout wordt ontdekt, hoe meer tijd het kost om deze op te lossen.

Vervolgens heb ik uitgelegd wat de methode *test-driven development* precies inhoudt en waarom ik denk dat dit bij Qualogy ingevoerd moet worden. De belangrijkste reden is dat ik vaak heb gehoord dat ontwikkelaars geen tijd hebben om unit testing toe te passen, vaak ook omdat het pas achteraf gebeurt.

Wel heb ik aangegeven dat test-driven development in het begin van het project soms moeilijk kan lijken, omdat het ontwikkelen dan nog langzaam gaat aangezien alle testscripts nog geschreven moeten worden. Echter, naarmate het project vordert, zal de reeds aanwezige testscripts ervoor zorgen dat je altijd flexibel bent en zeker kunt zijn van de kwaliteit van de applicatie.

Daarnaast heb ik continuous integration door middel van Jenkins gedemonstreerd. Ik heb laten zien hoe de dekkingsgraad een indicatie is voor de hoeveelheid aanwezige unittests en hoe testresultaten direct zichtbaar zijn. Tenslotte heb ik laten zien dat ook browsertesten geautomatiseerd kunnen worden door Selenium-testscripts te integreren in een unittestomgeving.

Het belangrijkste advies wat ik heb gegeven in de presentatie is:

- Mindset moet veranderen: ontwikkelaars moeten zich meer verantwoordelijk voelen voor het testen van de applicatie en dit niet zo gemakkelijk afschuiven op de testafdeling als dat nu gebeurt;
- Scrum correct toepassen en testers werkelijk deel uit laten maken van de ontwikkelteams;
- Test automation invoeren en test-driven development toepassen, door concrete doelstellingen per project vast te leggen;
- Continuous integration opzetten voor andere projecten.

Op het moment van presenteren was de documentatie (met name het implementatierapport) nog niet helemaal compleet, ik heb aangegeven dat ik dit thuis nog zou afronden. Na afloop van de presentatie gaf de directeur aan dat hij tevreden was met de resultaten en dat het duidelijk is hoe Qualogy Suriname hier zelfstandig mee verder kan gaan.

6. Evaluatie

In dit hoofdstuk blik ik terug op het verloop van het afstuderen, de uitdagingen waarmee ik te maken kreeg, de wijze van begeleiding en verbeterpunten ten aanzien van de resultaten.

6.1 Keuze afstudeeropdracht- en bedrijf

Bij het zoeken naar een geschikt bedrijf om bij te kunnen afstuderen, heb ik vooral gekeken naar grote softwarebedrijven. Het leek me interessant om te zien hoe een vergevorderd ontwikkelproces eruit ziet en hoe het is om in een volwassen ontwikkelteam werkzaam te zijn. Wat betreft de opdracht richtte ik me voornamelijk op web development, omdat mijn interesse daar ligt en ik daar al wat ervaring mee had. Verder wilde ik ook graag in het buitenland afstuderen voor mijn persoonlijke ontwikkeling en om mijn horizon te verbreden. Ik heb het in eerste instantie geprobeerd binnen Europa bij bedrijven als Google, Facebook, IBM en Microsoft. Toen dit niet lukte heb ik het bij wat kleinere software bedrijven geprobeerd in landen als Zwitserland, Frankrijk en Groot-Britannië.

Uiteindelijk kwam ik via Oracle bij Qualogy Suriname terecht. Hoewel Qualogy een groot bedrijf is, is de vestiging in Suriname nog redelijk jong (2007). Het ontwikkelproces bij de afdeling waar ik actief was, was daardoor niet dusdanig vergevorderd dan als ik had gehoopt. In plaats van te leren van collega's heb ik het meeste geleerd door middel van zelfstudie. Echter was het aspect van het werken in een team wel aanwezig, wat voor mij het belangrijkste was.

Ook betrof het een ander vakgebied, namelijk dat van *test automation*. Aangezien dit een voor mij nieuw vakgebied is, vormde dit een flinke uitdaging. Met name de onderzoeksfase heeft daardoor langer geduurd dan verwacht. Hulp van een collega uit Nederland heeft mij gelukkig geholpen met het bepalen van de juiste koers.

Ik heb geen spijt heb van mijn keuze voor Qualogy Suriname. Het was een prettig bedrijf om bij af te studeren. Echter had ik er meer uit kunnen halen wanneer men binnen het bedrijf meer kennis had omtrent de achtergrond van de opdracht, zodat de begeleiding wat gerichter kon zijn. Ook had het wellicht gemakkelijker geweest als ik voor een wat eenvoudigere opdracht had gekozen, bijvoorbeeld het ontwikkelen van een (web)applicatie. Dan zou ik echter niet zoveel nieuwe dingen hebben geleerd als nu het geval was.

6.2 Plan van aanpak

Het lastigste van het plan van aanpak was het SMART formuleren van de doelstelling. Mijn manager (de opdrachtgever) was duidelijk over de resultaten, namelijk tijdsbesparing en kwaliteitsverbetering, maar onduidelijk hoe deze resultaten meetbaar gemaakt konden worden. Hier is nooit echt duidelijkheid over geweest. Gelukkig bleek men achteraf tevreden te zijn met het resultaat.

Verder was het lastig om een planning te maken. Ik had me flink vergist in de onderzoeksfase, maar ook de implementatiefase bleek meer tijd te kosten dan gedacht. Ik denk wel dat ik met een wat strengere controle wat productiever en gerichter had kunnen werken.

6.3 Onderzoeksfase

De onderzoeksfase was nodig om het gehele bestaande testproces in kaart te brengen, tools te onderzoeken en met een passende oplossing te komen. In feite was dit de kern van de opdracht.

Over het algemeen vond ik het moeilijkste dat de gehele strategie voor de opdracht nog vastgelegd moest worden: vanwege de suggestie van Selenium ben ik eerst gaan zoeken naar record- en playback tools voor de testers, terwijl ik later zelf heb besloten om meer te richten op unit testing en continuous integration. Deze beslissingen heb ik geheel zelfstandig gemaakt zonder inbreng van de opdrachtgever of mentor, omdat zij hierin geen sturing gaven en ook nog niet veel wisten over test automation. Hierdoor heb ik heel veel zelfstudie moeten doen en later ook hulp moeten vragen aan een collega van Qualogy Nederland.

De interviews gingen vrij goed, maar - zoals meestal in zulke situaties - moet je echt weten wat je moet vragen en is het de kunst om door te vragen. Bovendien durfden sommige mensen soms niet open en bloot te zeggen wat ze ergens van vonden, daar moest je soms echt doorheen prikken. Ook was het soms nog zo dat er wat tegenstrijdige uitspraken werden gedaan.

Omdat ik zelf tot deze inzichten moest komen, ben ik relatief veel tijd kwijt geraakt aan uitzoekwerk en het beschrijven van de theorie. Indien het grote plaatje al eerder bekend zou zijn, zou ik veel meer tijd overhouden voor de implementatiefase.

Bij de toolselectie vond ik het moeilijk dat er zo ontzettend veel soorten tools zijn en dat er weinig concrete wensen waren vanuit de organisatie. Gelukkig heeft een collega uit Nederland mij geholpen bij het bepalen van de longlists en de afwegingen die leidden tot de shortlists. Wel had ik achteraf gezien minder tijd moeten besteden aan de record- en playback tools en de overige tools meteen moeten uitproberen bij een bepaald project. Eigenlijk was de gehele toolselectie te theoretisch uitgevoerd, het had absoluut iets praktischer aangepakt kunnen worden.

6.4 Implementatiefase

Bij het implementeren werd in het ontwikkelteam van het KvK-project geplaatst. Gedurende mijn werkzaamheden waren er slechts twee ontwikkelaars actief op dat project.

Allereerst heb ik met veel nieuwe technieken moeten leren werken: Maven, een build-tool voor Java-projecten, Groovy, een op Java gebaseerde programmeertaal, en Grails, een webapplicatieframework voor Groovy. Aangezien ik in het verleden al vaker met diverse frameworks heb gewerkt, had ik dit relatief snel onder de knie.

Verder heb ik Jenkins volledig opgezet en geconfigureerd voor het KvK-project. De belangrijkste uitdagingen heb ik al genoemd in hoofdstuk 4. Nogmaals: het was lastig dat ik alles zelf uit moest vinden. Achteraf gezien denk ik dat ik te zelfstandig heb gewerkt en collega's er meer bij had moeten betrekken.

6.5 Resultaten & oplevering

In deze paragraaf wil ik, kort samenvattend, de behaald resultaten evalueren. Gelet op het feit dat automatisch testen nog volledig in de kinderschoenen staat bij Qualogy Suriname, vind ik dat ik nog vrij veel heb bereikt. Men heeft bij de testafdeling met Selenium leren werken, zowel ontwikkelaars en ontwikkelaars zijn overtuigd geraakt van het belang van automatische unit- en browsertesten, en er is daadwerkelijk verbetering aangebracht bij een lopend project.

Toch enkele kritische punten:

- Het theoretische gedeelte (namelijk het onderzoek) en het praktijkgedeelte (de implementatie) hadden beter gecombineerd kunnen worden. Wat dat betreft had ik het beter iteratief kunnen aanpakken, zodat ik elke keer een stukje theorie onderzocht en dat meteen kon uitproberen. Ik geloof dat ik dan meer had kunnen bereiken.
- De theoretische basis vond ik erg sterk, maar de implementatie kon gewoon beter. Zo had ik liever dat collega's al vroegtijdig zelfstandig met de unittesten konden werken, zodat ik de resultaten echt had kunnen bekijken.

De belangrijkste vraag is echter: is de doelstelling behaald? Wat betreft de tijdswinst absoluut zeker. In de volgende tabel staat bijvoorbeeld een (vereenvoudigde) weergave van de tijdswinst die behaald is met het werken met Selenium IDE bij de frontoffice van het KvK-project.

Aantal formulieren	22
Aantal testgevallen	77
Handmatig testen	
Aantal min/testgeval	15-45
Totaal aantal minuten	1155-3465
Automatisch testen	
Aantal min/testgeval	5-15
opstellen script voor formulier: aantal minuten (eenmalig) per formulier	5-15
runnen/verbeteren van script: per formulier	30
Totaal aantal minuten eenmalig (per formulier)	35-45
Totaal aantal minuten eenmalig (aantal formulieren * totaal aantal minuten eenmalig)	770-990
Totaal aantal minuten (= aantal testgevallen * aantal min/testgeval)	385-1155
Total aantal minuten (eerste test)	Max 2145
Totaal aantal minuten overige testen	Max 1815
Maximaal verschil	**3465-1815 = 1650 min (= 27,5 uur)**

Wat betreft de kwaliteitsverbetering, dit is uiteraard moeilijker meetbaar. De traject was te kort om dit duidelijk te kunnen meten. De dekkingsgraad van de unittesting is bij één project wel verbeterd, maar dit betekent natuurlijk niet direct een kwaliteitsverbetering. De tijd zal dit moeten leren. Ik heb in ieder geval de juiste handvatten aangereikt waarmee Qualogy in de toekomst op kan voortborduren.

6.5 Begeleiding

De begeleiding tijdens het afstudeerproject bestond voornamelijk uit het rapporteren van de voortgang en het bewaken van de planning. Ongeveer één keer per 2 á 3 weken was er een meeting waar mijn mentor en mijn manager bij aanwezig waren. Tijdens de meetings moest ik vertellen wat ik had gedaan en of we nog op planning waren. Eventuele actiepunten werden besproken en er kwam feedback op eventuele opgeleverde producten.

Buiten de meetings kreeg ik veel zelfstandigheid bij het uitvoeren van de werkzaamheden. Dit was aan de ene kant erg prettig, omdat ik zelf alle keuzes mocht maken en zelf ook mijn tijd mocht indelen. Aan de andere kant was dit ook best lastig, omdat ik alles zelf moest organiseren. Zo moesten er bijvoorbeeld demo's en presentaties gegeven worden, maar aangezien ik hierin erg terughoudend ben geweest kwam hiervan pas op het laatste moment. Het lastigste vond ik om te bepalen wie hierbij aanwezig moesten zijn en het inplannen van een geschikt moment.

Achteraf gezien had ik mijn vrijheid en zelfstandigheid beter kunnen benutten. Bijvoorbeeld door meer samen te werken met één of meer ontwikkelaars, en de demo's en presentaties eerder te organiseren. Dit is echter met de kennis achteraf gemakkelijker gezegd dan gedaan.

6.6 Beroepstaken

Voorafgaand aan het afstudeertraject had ik de volgende beroepstaken geselecteerd om mijn competenties te kunnen demonstreren:

1. Selecteren methoden, technieken en tools (niveau 3)
2. Selecteren van standaardsoftware (niveau 3)
3. Ontwerpen softwarearchitectuur (niveau 3)
4. Initiëren en plannen van het testproces (niveau 3)

De beroepstaken zijn allemaal minimaal op niveau 3 uitgevoerd, omdat deze ofwel *lastig-zelfstandig* waren, ofwel *simpel-sturend*. De meeste taken zijn zelfstandig uitgevoerd (zie paragraaf 6.5), bij andere taken heb ik collega's aangestuurd.

6.6.1 Selecteren methoden, technieken en tools (1.1) + Selecteren van standaardsoftware (1.3)

Deze beroepstaken is vormen de kern van mijn afstudeeropdracht. Vrijwel het gehele traject bestond namelijk uit het onderzoeken van geschikte methodes voor (geautomatiseerd) testen en het selecteren van de juiste tools hiervoor.

Allereerst heb ik geautomatiseerd testen door middel van *record- en playback* tools onderzocht. Ik heb tijdens de toolselectie met een aantal van dergelijke tools gewerkt en heb de tool Selenium

geïntroduceerd op de testafdeling. De beperkingen van dergelijke tools werden echter ook duidelijk zichtbaar.

Daarop ben ik in de richting van test-driven development gegaan en ben me meer gaan bezighouden met unit testing en geautomatiseerde browsertesten. Hierbij heb ik ook behaviour driven development onderzocht, maar heb besloten om dit niet in te voeren binnen Qualogy Suriname omdat men hier (nog) niet aan toe was.

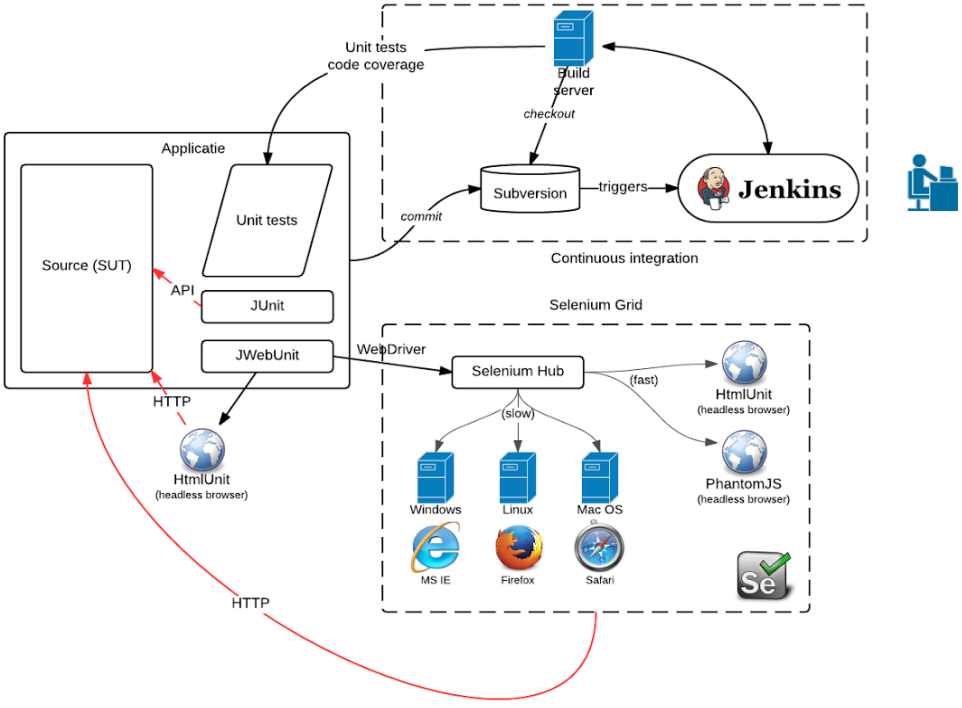
Verder heb ik ook continuous integration ingevoerd en hiervoor een Jenkins-server opgezet en geconfigureerd.

Bij het bepalen van de methodes, technieken en tools heb ik rekening gehouden met de volgende factoren:

- specifieke wensen van de organisatie (bijv. gebruik van open-source tools)
- het gebruik van Agile methodes zoals scrum
- de aanwezige expertise van de testers en ontwikkelaars
- de gewenste kwaliteit en dekkingsgraad

6.6.2 Ontwerpen softwarearchitectuur (3.1)

Voor het project heb ik een test framework opgezet waarbij testers en ontwikkelaars met elkaar samenwerken, gebruik makend van diverse tools. Bovendien is er een Selenium Grid configuratie ontworpen, zodat er vanuit Jenkins automatisch getest kan worden op diverse besturingssysteem/browser combinaties. Zie paragraaf 4.4 en figuur 6.1.



Figuur 6.1 - De architectuur van continuous integration i.c.m. een Selenium Grid-configuratie

6.6.3 Initiëren en plannen van het testproces (3.4)

Tijdens de implementatie heb ik meegewerkt aan het testproces van het KvK-project. Ik heb hierbij unit- en unitintegratietesten geschreven in een voor de organisatie nieuw testframework, namelijk Spock. Ook zijn er tools gebruikt voor het meten van de dekkingsgraad. Tenslotte heb ik ook collega's getraind in het gebruik hiervan.

6.7 Conclusie

Ik kan terugblikken op een erg leuke maar vooral ook leerzame afstudeerperiode bij Qualogy Suriname. De opdracht was een stuk pittiger dan gedacht, maar persoonlijk heb ik er heel veel aan gehad. Niet alleen op technisch vlak heb ik veel geleerd, ook heb ik weer een stuk van het bedrijfsleven mogen proeven. Het werken bij een wat groter bedrijf trekt me nu nog meer aan dan voor het afstuderen.

Hoewel ik er wel nog meer uit had willen halen, kan ik toch met tevredenheid terugblikken op een succesvolle opdracht. Ik hoop dat Qualogy Suriname veel profijt zal hebben van het opgeleverde advies en was de implementatie voldoende om zelfstandig ermee verder te gaan.

Tenslotte kan ik Qualogy van harte aanbevelen als afstudeerbedrijf bij andere studenten! Het is een prettige werkgever met flexibele arbeidsvoorwaarden en de sfeer is erg collegiaal en hartelijk.

Literatuurlijst

Internetbronnen

Scott, Alister (31 januari 2012). *Introducing the software testing ice-cream cone (anti-pattern)*.
http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/

Reminnyi, Oleksandr (2 augustus 2013). *Functional GUI Testing Automation Patterns*.
http://www.infoq.com/articles/gui-automation-patterns

Selenium Documentation
http://www.seleniumhq.org/docs/

Wikipedia, *Test automation*.
http://en.wikipedia.org/wiki/Test_automation

Wikipedia, *List of GUI testing tools*
http://en.wikipedia.org/wiki/List_of_GUI_testing_tools

Literatuur

Tim Koomen, Leo van der Aalst, Bart Broekman, Michien Vroon (2014). *TMap NEXT voor resultaatgericht testen*. Groningen: LINE UP boek en media B.V.

Leo van der Aalst, Cecile Davis (2012). *TMap NEXT in scrum - Effectief testen in agile projecten.* Groningen: LINE UP boek en media B.V.

Paul Swartout (2012). *Continuous Delivery and DevOps: A Quickstart Guide*. Birmingham (UK): Packt Publishing Ltd.

Verklarende woordenlijst

Hieronder staat een verklarende woordenlijst. De woorden staan niet in alfabetische volgorde, maar in volgorde van relevantie.

Testen	Activiteiten die uitgevoerd worden om één of meer kenmerken van een product, proces of dienst vast te stellen volgens een gespecificeerde procedure. (volgens TMap NEXT®)
Ontwikkeltesten	Testen die doorgaans door de ontwikkelaars worden uitgevoerd. Hierbij vormen de technische specificaties de belangrijkste testbasis en is er inzicht in de werking van het systeem (white box). Hierbij wordt op laag niveau getest, zoals de zogenaamde unittests en unit-integratietests.
Systeemtesten	Deze worden veelal door speciale testers uitgevoerd. Hierbij vormt met name het functioneel ontwerp de testbasis. Hierbij wordt er op een hoger niveau getest, zoals de end-to-end tests.
Acceptatietesten	Deze worden uitgevoerd door de accepterende partij. Hierbij speelt vooral de verwachting een belangrijke rol. Een voorbeeld is een gebruikersacceptatietest, die door de eindgebruiker van het systeem wordt uitgevoerd.
Functionele testen	Testen die zich met functionele eisen bezighouden, in tegenstelling tot niet-functionele eisen zoals performance en usability.
Regressietesten	Met regressietesten wordt gecontroleerd of de niet aangepaste onderdelen van een applicatie nog steeds juist werken.
Browsertesten	Zie: webtesten.
Webtesten	Dit zijn functionele testen voor webapplicaties die, in tegenstelling tot bijvoorbeeld unit testing, vanuit een webbrowser met het systeem interacteren. Door middel van browser automation kan dit proces worden geautomatiseerd.
Unittesten	Een vorm van testen waarbij ontwikkelaars een testscript schrijven waarbij bepaalde onderdelen (units, bijv. een bepaalde klasse) los worden getest.
Continuous integration	Een verzameling van best practices waarbij meerdere ontwikkelaars meerdere keren per dag hun code centraal samenvoegen, wat vervolgens automatisch wordt gebuild en getest en waarop de resultaten direct zichtbaar zijn. Zie ook: Jenkins.
Test-driven development (TDD)	Een methode waarbij het schrijven van testen vóór het bouwen van de software plaatsvindt, om o.a. het testen te stimuleren.

Behaviour-driven development (BDD)	Een specifieke vorm van BDD waarbij de acceptatietesten d.m.v. user stories het uitgangspunt vormen.
Dekkingsgraad	De hoeveelheid code die gedekt wordt door testscripts. Kan uitgedrukt worden in onder andere het percentage regels, methods, classes en packages.
Building	Het bouwproces van een softwareapplicatie, waarbij afhankelijkheden worden geïnstalleerd, code wordt gecompileerd en eventuele tests worden uitgevoerd.
Maven	Een build- en package management tool voor Java-projecten.
Grails	Een framework voor webapplicaties, gebaseerd op de Java-programmeertaal Groovy.
Spock	Een testframework voor Java.
Jenkins	Een tool waarmee men continuous integration kan opzetten.

Bijlage A - Plan van aanpak

Bijlage B - Interviews

Bijlage C - Beschrijving bestaande situatie

Bijlage D - Conceptadvies 10 september

Bijlage E - Toolselectierapport

Bijlage F - Adviesrapport

Bijlage G - Evaluatieformulier