QUALOGY

PLAN VAN AANPAK

v.1.0

Afstudeeropdracht bij Qualogy Suriname

Door Dirk Luijk • 21 augustus 2014

Inhoudsopgave

Inleiding

In dit document wordt het plan van aanpak beschreven van de afstudeeropdracht van Dirk Luijk bij Qualogy Suriname.

De afstudeeropdracht vindt plaats bij de Surinaamse vestiging van het Nederlandse bedrijf Qualogy, gevestigd in Paramaribo. Qualogy levert maatwerk in Oracle- en Java- applicatieontwikkeling, Oracle E-Business, database- en applicatiebeheer, Business Intelligence, Agile Consultancy en Web 2.0 development.

Dirk Luijk studeert Informatica aan de Haagse Hogeschool en is voor de afstudeeropdracht naar Suriname afgereist om daar in een periode van vier maanden bij Qualogy op de testafdeling werkzaam te zijn. Met voornamelijk werkervaring als web developer hoop hij met zijn technische achtergrond het testproces zoveel mogelijk te automatiseren.

In het plan van aanpak wordt er allereerst stilgestaan bij de probleemstelling. Hierbij wordt een korte samenvatting gegeven van de huidige situatie en welke knelpunten zich daarbij voordoen en wordt er een doelstelling geformuleerd. Daarna volgt een omschrijving van de opdracht zelf, en wordt stilgestaan bij de activiteiten, producten en planning, maar ook bij zaken als risico- en kwaliteitsmanagement.

1. Probleemstelling

In dit hoofdstuk volgt een beschrijving van de huidige situatie, welke problemen zich voordoen en de doelstelling voor de afstudeeropdracht die daaruit volgt.

1.1 Situatiebeschrijving

Bij Qualogy Suriname zijn er o.a. de volgende afdelingen:

- Java-afdeling (vier scrum teams)
- Oracle/Apex-afdeling
- Quality Control (testafdeling)

> *Meer informatie over de organisatiestructuur is te vinden in het bijgeleverde organogram.*

De meeste projecten worden als volgt doorlopen:

1. Het betreffende ontwikkelteam stelt de requirements en een functioneel ontwerp op;
2. Het testteam maakt op basis hiervan de testontwerpen;
3. Zodra het ontwikkelteam de software aanlevert, wordt deze door het testteam getest;
4. Bij onvoldoende kwaliteit wordt het teruggestuurd naar de ontwikkelafdeling, en worden de bevindingen verwerkt;
5. Bij voldoende kwaliteit kan het worden opgeleverd naar de klant. Deze neemt het na acceptatie in productie.

Verder wordt er binnenkort bij het Oracle-ontwikkelteam overgestapt van de watervalmethode naar Scrum.

> *Meer informatie over de huidige situatie wordt tijdens het project middels onderzoek inzichtelijk gemaakt, zie ook paragraaf 2.3 en 2.4.*

1.2 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 testen en regressietesten.
- 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.

1.3 Doelstelling

De oplossing en daarmee ook de doelstelling van het project is, net als de probleemstelling, tweeledig:

- 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* 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.

Om de doelstelling zo duidelijk, concreet en eenduidig mogelijk te maken, wordt deze hieronder getoetst aan het SMART-principe:

Specifiek	Dirk Luijk zal binnen de test- en ontwikkelafdelingen van Qualogy Suriname de ontwikkel- en systeemtesten automatiseren 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.
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.

2. Beschrijving opdracht

In dit hoofdstuk wordt de afstudeeropdracht beschreven. Allereerst wordt er een samenvatting gegeven van de uit te voeren activiteiten (paragraaf 2.1), vervolgens wordt er aangegeven wat er wel en niet binnen de scope van de opdracht valt (paragraaf 2.2). Tenslotte sta ik stil bij de activiteiten die uitgevoerd worden en volgens welke planning (paragraaf 2.3), de producten die daarbij opgeleverd worden en hoe omgegaan wordt met risico's (paragraaf 2.5). Ook wordt er ingegaan op het kwaliteitsmanagement (paragraaf 2.6) en worden de randvoorwaarden genoemd (paragraaf 2.7).

2.1 Samenvatting

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.

Vervolgens wordt er een adviesrapport opgesteld waarin er oplossingen wordt aangedragen voor de geconstateerde problemen. Hierbij zullen er ook tools ingeschakeld worden, dus zal er een uitgebreide toolselectie aan vooraf gaan.

Daarna zullen de wijzigingen geimplementeerd worden, waarbij de tools aangeschaft, geïnstalleerd en geconfigureerd worden. Vervolgens zullen er trainingen gegeven worden om de gebruikers te leren werken met de nieuwe software.

2.2 Afbakening

Alleen de volgende testen zullen worden geautomatiseerd:

- Ontwikkeltesten: geautomatiseerde unittests
- Systeemtesten:
    - functionele (regressie)testen
    - formulieren testen

Verder wordt er gekeken naar een manier om requirements op een uniforme wijze op te stellen en of tools hierbij kunnen helpen.

2.3 Activiteiten en planning

In de volgende matrix staan alle activiteiten, de bijbehorende deliverables (producten) en afhankelijkheden. Verder zijn er tussendoor meetmomenten, dan vindt er overleg plaats met de opdrachtgever.

Periode / datum	Activiteit(en)	Producten	Benodigdheden
week 1 18 aug t/m 22 aug	*Opstellen plan van aanpak*	Plan van aanpak	Input van Ray en Purcy
27 augustus	Overleg plan van aanpak		
week 2 - 3 25 aug t/m 5 sept	*Onderzoek naar bestaande situatie* *Verzamelen informatie*	Onderzoeksrapport huidige situatie	Interviews met testafdeling Interviews met Java/Apex Toegang tot JIRA
week 4 8 sep - 10 sep	*Formuleren eenvoudig verbeteringsvoorstel* *Selectie kandidaten voor toolselectie*	Verbeteringsvoorstel met kandidaten toolselectie	Randvoorwaarden en selectiecriteria
10 september	Overleg verbetervoorstel		
week 4 - 6 11 sep - 26 sep	*Uitvoeren toolselectie*		
week 7 29 sep - 3 okt	*Opstellen adviesrapport*	Adviesrapport (concept)	Feedback op adviesrapport van Ray en Purcy
week 8 6 okt - 10 okt	*Verwerken van feedback op adviesrapport*	Adviesrapport (definitief)	Akkoord op adviesrapport
nader te bepalen	*Presentatie adviesrapport*		
week 9 - 13 13 okt - 14 nov	*Tools implementeren en configureren*		Opdracht management voor aanschaf software Assistentie van systeembeheer bij inrichting tools
week 14 - 16 17 nov - 5 dec	*Trainingen geven*	Trainingsmateriaal en handleidingen	
nader te bepalen	*Evaluatie met opdrachtgever*		
week 17 - 18 8 dec - 19 dec	*Afronden project, documentatie en stageverslag*		

2.4 Producten

De volgende producten worden volgens het reeds beschreven tijdpad opgeleverd:

- **Plan van aanpak**
  Dit document.
- **Onderzoeksrapport huidige situatie**
  Hierin wordt op basis van o.a. interviews beschreven hoe de huidige situatie eruit ziet en wordt de achtergrond van het probleem duidelijk gemaakt.
- **Verbeteringsvoorstel (conceptversie adviesrapport)**
  Hierin wordt concreet benoemd hoe de in het onderzoeksrapport aangetoonde problemen opgelost gaan worden, en waar er tools ingezet gaan worden. Het voorstel bevat verder een aantal randvoorwaarden, selectiecriteria en een lijst met kandidaten voor de toolselectie. Dit is in principe een eerste opzet van het adviesrapport.
- **Adviesrapport**
  Dit rapport bevat o.a. de resultaten van de toolselectie en bevat een conclusie omtrent welke tools ingezet moeten gaan worden. Tevens wordt beschreven welke andere aanpassingen en maatregelen uitgevoerd moeten worden om het testen te automatiseren.
- **Implementatierapport**
  Hierin wordt beschreven hoe de gekozen tools zijn geïmplementeerd en hoe ermee gewerkt wordt, zodat de opgeleverde oplossing tastbaar wordt voor de organisatie.
- **Richtlijnen voor unittests / continuous integration**
  Hierin wordt beschreven hoe ontwikkelaars ervoor kunnen zorgen dat de kwaliteit van het door hun geleverde systeem van voldoende kwaliteit is om opgeleverd te kunnen worden aan de testafdeling. Dit omvat o.a. instructies omtrent unit testen (bijv. minimale dekkingsgraad) en hoe de build server ingericht moet worden.
- **Trainingsmateriaal / handleidingen**
  Aanvullend materiaal zoals presentaties en handleidingen voor specifieke taken.

2.5 Risicomanagement

Risico	Hoe te voorkomen	Hoe op te lossen
Er is (te) weinig animo bij de ontwikkelteams voor geautomatiseerd unittesten.	Er wordt een presentatie gegeven waarin op een aansprekende manier alle voordelen van unittesten worden benoemd.	In het begin een niet al te streng beleid voeren, en/of een specifieke groep toewijzen voor unittests.
De gekozen tools zijn niet effectief.	De tools eerst goed uitproberen, gebruik makend van probeerversies. Gebruikers al in een vroeg stadium ermee laten werken, zodat tijdig kan worden ingegrepen.	Een definitieve oplossing vooraf is niet denkbaar. Er wordt uitgezocht waar het probleem aan ligt, en vroegtijdig besproken met het management.
De opdracht blijkt niet omvangrijk genoeg (voor de examinatoren).	Het plan van aanpak wordt opgestuurd naar de eerste examinator (begeleidende docent) voor feedback en goedkeuring.	Er wordt met de begeleidende docent en met het management besproken hoe de opdracht uitgebreid kan worden.
Mismatch in verwachtingen	Frequent statusoverleg, demo's	

2.6 Kwaliteitsmanagement

De opgeleverde documentatie moet van dermate kwaliteit zijn, dat...

- … iemand die op geen enkele wijze betrokken is geweest bij het project, in staat moet zijn te begrijpen wat is er is uitgevoerd, waarom, hoe en wanneer.
- … zowel management, ontwikkelaars als testers begrijpen hoe de nieuwe situatie werkt, welke richtlijnen en werkwijze hierbij van toepassing zijn en hoe de tools hierin functioneren.
- … er voor het management een korte samenvatting wordt opgenomen in de opgeleverde rapporten, zodat deze snel kunnen zien wat er op business niveau zal veranderen / is veranderd.
- … het management begrijpt op welke manier er tijd bespaart wordt en hoe de veranderingen zich terugverdienen.

De documentatie hoeft niet van dermate kwaliteit te zijn, dat ontwikkelaars en testers uitsluitend op basis hiervan met de tools kunnen leren werken. Er wordt ondersteunend materiaal opgeleverd, maar de nieuwe software moet zelf voldoende documentatie (zoals handleidingen) bevatten om ermee te kunnen werken.

2.7 Randvoorwaarden

Voor het uitvoeren van de opdracht heeft de opdrachtnemer de volgende zaken nodig van de opdrachtgever:

- Toegang tot projectdocumentatie, wanneer dit gevraagd wordt;
- Toegang tot het projectmanagementsysteem, JIRA;
- De mogelijkheid om mensen van de betrokken afdelingen te interviewen;
- Beschikbaarheid van een vaste bedrijfsmentor voor advies en feedback, van ten minste drie dagen per week en één keer per dag.

Interview Java-afdeling

Donderdag 5 september 2014

Aanwezig	Jozua Herfst
Interviewer	Dirk Luijk

Inleiding

Op donderdag 5 september is er een interview geweest met Jozua Herfst, Java Developer bij Qualogy Suriname. Het doel van het interview was om informatie in te winnen over de stand van zaken op de Java-ontwikkelafdeling. De volgende onderwerpen kwamen hierbij aan bod:

1. De organisatie op de afdeling
2. De manier van werken / ontwikkelmethodes
3. Infrastructuur en gebruikte tools
4. Draagvlak voor unittesting en continuous integration

Hoeveel mensen werken er op de afdeling en hoeveel projecten worden er hier tegelijk uitgevoerd?
De Java-afdeling is opgebouwd uit verschillende teams. Ons team is zes man groot en werkt momenteel aan het KvK project. Verder heb je de projectteams GLIS, UTS, UCC en Telesur. De laatste twee teams werken op locatie bij de klant.

Welke ontwikkelmethode gebruiken jullie?
Alle genoemde Java-teams werken volgens scrum. Ons team werkte eerst met sprints van één week, maar dit is veranderd naar twee weken omdat het testteam ook tijd nodig heeft om het product binnen de sprint te kunnen testen.

Hoe ziet de documentatie van een project eruit? Maken jullie gebruik van een standaard format?
Onze documentatie bevat een functioneel en een technisch ontwerp. Het functioneel ontwerp wordt éénmalig gemaakt en bevat alleen een heel globale beschrijving (afbakening) van het project. Het technisch ontwerp bevat UML diagrammen zoals de sequence- en activity diagrams. Voor de documentatie gebruiken we Google Drive. Verder worden er volgens de scrum methodiek *user stories* beschreven. Hiermee worden requirements onderbouwd in subtaken. Deze worden bijgehouden in JIRA Agile.

Er is helaas nog geen standaard format. We zouden graag een template willen hebben voor het functioneel en technisch ontwerp, zodat deze ook consistent zijn met die van het Oracle-team. We hebben hier wel om gevraagd bij het management, maar dit is tot op heden nog niet beschikbaar.

Hoe uitgebreid is jullie documentatie?
Onze documentatie is vrij dun. Dit heeft meerdere oorzaken. Allereerst bestaat ons scrum team nagenoeg uit developers, die vaak geen tijd krijgen om aan documentatie te werken vanwege de tijdsdruk die de sprints met zich meebrengen. Daarnaast is veel informatie in onze hoofden opgeslagen en zijn de requirements vanwege de standup-meetings bij iedereen bekend. De onduidelijkheid zit hem daarom meestal in de kleine dingen.

Welke schematechnieken passen jullie meestal toe?
Wij maken gebruik van UML in onze documentatie, eigenlijk voornamelijk in het technisch ontwerp. Schema's zoals *sequence diagrams* en *activity diagrams* helpen de ontwikkelaars binnen het team een situatie te begrijpen.

Op welke manier leggen jullie requirements vast?
We werken met user stories. Uit de user stories komen taken voort, welke door middel van functiepunten worden ingepland.

Ik heb begrepen dat jullie gebruik maken van unittests?
Jazeker, we gebruiken het al aardig. We maken daarbij gebruik van de testframeworks JUnit, Mockito en Easymock. Unittesten vinden we handig omdat we dan de klant kunnen garanderen dat iets echt werkt. We proberen testgedreven te ontwikkelen (TDD), maar soms gebeurt het dat we de tests achteraf schrijven vanwege de tijdsdruk van een sprint.

En hoe zit het met versiebeheer?
Wij maken gebruik van SVN.

Hebben jullie wel eens nagedacht over *continuous integration*?
Jazeker**,** er is wel eens gewerkt met Jenkins. Ons team doet er momenteel niet zoveel mee, maar ik weet dat een ander projectteam (UTS) met Jenkins werkt voor deployment.

Interview Java-afdeling

Maandag 8 september 2014

Aanwezig	Rogér Pique Ray Amatsoleman
Interviewer	Dirk Luijk

Op maandag 8 september is er een interview geweest met Rogér Pique, Java Developer, en Ray Amatsoleman, Test Engineer. Het doel van het interview was om informatie in te winnen over de stand van zaken op de Java-ontwikkelafdeling, als aanvulling op het eerder gehouden interview met Jozua Herfst.

Aan welk project werken jullie momenteel en hoe groot is jullie projectgroep?
Wij werken momenteel aan UTS (telecomprovider op Curaçao). Onze projectgroep bestaat uit zo'n 9 ontwikkelaars en één tester.

Welke ontwikkelmethode passen jullie toe?
Wij werken nu al enkele jaren volgens scrum. Wij hanteren sprints van 1 week lang.

Maken jullie gebruik van unittests?
Jazeker. Het is niet zo dat we direct met de tests beginnen (volgens *test-driven development*) maar we proberen wel zoveel mogelijk met unittests te dekken. Door de tijdsdruk lukt dit echter lang niet altijd.

Hoe groot denk je dat de dekkingsgraad van jullie unittests is?
Hiervan hebben we geen harde cijfers. We schatten dat dit ongeveer 50-60% is.

Hoe ziet jullie documentatie eruit?
Ons functioneel ontwerp bevat voornamelijk user stories, deze slaan we op in JIRA Agile. Verder werken we ook wel met Excel. Voor het technisch ontwerp werken we nog wel eens met stroomdiagrammen. Aangezien de klant ook inzage heeft in het technisch ontwerp, kunnen we geen ingewikkelde schematechnieken toepassen omdat de klant dit anders mogelijk niet begrijpt.

Hoe vaak is er contact met de klant?
Dagelijks. Ons team heeft, zoals scrum dat noemt, een *product owner*. Deze is verantwoordelijk voor het in kaart brengen van alle wensen van alle stakeholders en vertegenwoordigt hen binnen de projectgroep.

Wat gebruiken jullie voor versiebeheer?
Wij maken gebruik van SVN. Elke ontwikkelaar doet per dag zo'n drie tot vier commits.

Maken jullie gebruik van *continuous integration*?
Jazeker. Wij maken nu al zo'n 2 jaar gebruik van *Jenkins*. Eerst gebruikten we het alleen voor het automatisch builden van de software en het uitvoeren van de unittests, later ook voor deployment. Momenteel kunnen we de software namelijk binnen een paar klikken uitrollen naar de verschillende omgevingen zoals test, acceptatie en deployment (volgens OTAP).

Wat zou de reden zijn dat andere teams er nog geen gebruik van maken?
Mogelijk zijn zij er nog niet aan toe, wij hebben daar niet zo'n zicht op. Wij kunnen het gebruik van Jenkins in ieder geval de andere teams van harte aanbevelen. Het is, met behulp van de juiste plugins, ook redelijk makkelijk op te zetten.

Bedankt voor jullie tijd.

Interview Oracle-afdeling

Woensdag 4 september 2014

Aanwezig	Sullivan Kromosoeto
Interviewer	Dirk Luijk

Inleiding

Op woensdag 3 september is er een interview geweest met Sullivan Kromosoeto, Oracle Lead Developer op de Oracle-ontwikkelafdeling bij Qualogy Suriname.

Het doel van het interview was om informatie in te winnen over de stand van zaken op de Oracle-ontwikkelafdeling. De volgende onderwerpen kwamen hierbij aan bod:

1. De organisatie op de afdeling
2. De manier van werken / ontwikkelmethodes
3. Infrastructuur en gebruikte tools
4. Draagvlak voor unittesting en continuous integration

Hoeveel mensen werken er op de afdeling en hoeveel projecten worden er hier tegelijk uitgevoerd?
Er werken hier 7 mensen en er lopen op dit moment vijf projecten. Normaal lopen er zo'n drie tot zes projecten. Er zijn hier enkele lead developers, zoals Dennis en ik, die anderen aansturen binnen een project. Hoofd van de afdeling is Frank van der Proeg.

Welke ontwikkelmethode gebruiken jullie?
De ontwikkelmethode bevat typische kenmerken van de watervalmethode: op basis van een vaste prijs en een afgesproken doorlooptijd wordt er een planning gemaakt en we gaan pas aan de slag wanneer alle requirements bekend zijn. We gaan binnenkort experimenteren met een functiepuntenanalyse.

Hoe ziet de documentatie van een project eruit? Maken jullie gebruik van een standard format?
We stellen een functioneel ontwerp en een technisch ontwerp op. We gebruiken meestal het functioneel- en technisch ontwerp van het vorige project als uitgangspunt, maar hebben niet echt een specifiek format.

Ik heb begrepen van de testafdeling dat de documentatie van jullie afdeling vrij veel tekst bevat, en weinig schema's en modellen. Is dat zo?
Ja, dat klopt wel. We zouden best wel meer schematechnieken zoals UML willen gebruiken, maar dan lopen we het risico dat de opdrachtgever de schema's niet begrijpt. Bovendien krijg je dan ook een discussie over de vraag hoe schema's eruit moeten zien en hoe ze gelezen

moeten worden. Daarom houden we het heel simpel en gebruiken we voornamelijk teksten en tabellen.

Hoe weet de testafdeling bijvoorbeeld welke velden verplicht zijn en hoe welke validatieregels van toepassing zijn?
Het datamodel beschrijven we in het technisch ontwerp, meestal in eenvoudige tabelvorm. De testafdeling heeft in principe ook inzage in het technisch ontwerp.

Op welke manier leggen jullie requirements vast?
In principe zijn sommige requirements al beschreven in de workflows / use cases in het functioneel ontwerp. Andere requirements houden we gewoon bij in een lijst.

Hoe beoordelen jullie of een product gereed is om getest te worden?
Het product is in orde wanneer alle issues, die tijdens de vorige test aan het licht kwamen, zijn opgelost. Deze issues worden bijgehouden in JIRA.

Jullie gebruiken dus voornamelijk de gerapporteerde issues om de kwaliteit vast te stellen. Maar hoe zit het dan bij de eerste release, wanneer er nog geen issues zijn?
Nou, we testen uiteraard zelf ook onze code. Niet zo uitvoerig als de testafdeling dit doet, maar iedereen moet toch wel zijn/haar onderdeel controleren en pas wanneer iedereen aangeeft dat alles werkt, wordt het opgeleverd.

Hoe verloopt de interactie met de testafdeling precies? Hoeveel correctierondes zijn er doorgaans?
Het aantal correctierondes is uiteraard afhankelijk van de omvang en complexiteit van een project. Maar wat ook belangrijk is: het hangt ook af van kwaliteit van de feedback.

Hoe bedoel je dat, is de feedback van de testafdeling dan niet altijd even goed?
Nee, niet altijd. Soms is een issue te vaag omschreven en is bijvoorbeeld niet bekend hoe het probleem gereproduceerd kan worden.

Met welke software werken jullie?
We ontwikkelen hier in Oracle en Apex. Oracle is de database waar we onze business logica in implementeren. Apex is de applicatielaag en frontend. Voor Oracle werken we met SQL Developer en PL/SQL Developer en voor Apex werken we in de webomgeving.

Op basis waarvan wordt eigenlijk besloten of een project bij Java of Oracle terecht komt?
Dit wordt bepaald door het management en is afhankelijk van diverse factoren. Bijvoorbeeld wat voor type project het is, welke vereisten er zijn, bij welk team er genoeg capaciteit is, maar het gebeurt ook nog wel eens dat de klant geen Oracle database wil. Dan gaat het project naar het Java-team.

Passen jullie ook unittesten toe?
Nee, nog niet. Persoonlijk zou ik dit graag zien, maar momenteel is er nog te weinig draagvlak in het team. Het is overigens goed mogelijk in PL/SQL Developer. Maar de meeste mensen hier zien het nut er niet echt van in en hebben ook niet echt een motivatie om het te gebruiken.

Wat voor versiebeheersoftware gebruiken jullie?
We gebruiken momenteel geen versiebeheer. Er is in het verleden wel geëxperimenteerd met Subversion (SVN), maar daar is het bij gebleven.

Zou *continuous integration* handig zijn binnen jullie team?
Voor wat betreft het unittesten wel, omdat de kwaliteit dat inzichtelijker wordt. Maar wat betreft het gebruiken van versiebeheer en het dagelijks inchecken, dat lijkt mij overbodig. We werken met slechts twee á drie man aan een project en werken gewoon tegelijk in dezelfde omgeving. Daarbij gebeurt het bijna nooit dat er een conflict optreedt bij het opslaan van onze code, omdat we heel duidelijk afspreken wie waaraan werkt. Bovendien wordt er door de DBA (database administrator) minimaal één keer per dag een backup gemaakt van de diverse omgevingen.

Hoe ga je dan te werk als er een probleem optreedt in de productieomgeving?
Verder werken we volgens het OTAP principe (ontwikkel-test-acceptatie-productie), dus kunnen we heel gemakkelijk de laatste versie uit de acceptatie- of testomgeving halen waarin we dan de fix kunnen implementeren. Als de ontwikkelversie afwijkt van de productieversie, moeten we de fix uiteraard ook daar nog in implementeren.

Zijn er nog andere knelpunten in het testproces?
Ja, soms vind ik dat er teveel aandacht wordt besteed aan minder belangrijke functionaliteit. De prioritering tijdens het testen is soms niet logisch. Een voorbeeld: soms wordt er veel tijd besteed aan het testen van validatie van formulieren, zoals teveel karakters, terwijl andere onderdelen zoals financiële berekeningen veel meer aandacht verdienen.

Verder - ik gaf het zojuist al aan - zijn de vastgelegde issues in JIRA soms van slechte kwaliteit. Het zou mooi zijn als dit dusdanig wordt gestandaardiseerd dat er standaard bij wordt vermeld wat het verschil tussen de verwachte en daadwerkelijke uitvoer is, hoe het probleem gereproduceerd kan worden, eventueel aangevuld met een screenshot. Bovendien zou de lijst met issues ook beter gecategoriseerd kunnen worden.

Bedankt voor je tijd!

Interview testafdeling

Maandag 18 augustus 2014

Inleiding

Op maandag 18 augustus is er een kort interview geweest met Kim Ilahi en Gail Djaspan van de testafdeling bij Qualogy Suriname. Ook Purcy Sweeb, delivery manager, was aanwezig.

Het doel van het interview was om genoeg informatie te verzamelen om een plan van aanpak te kunnen maken voor de afstudeeropdracht. De volgende onderwerpen kwamen aan bod:

1. Algemene informatie over het bedrijf
2. Hoe het gehele testproces op dit moment is ingericht

Aanwezig: Kim Ilahi, Gail Djaspan, Purcy Sweeb
Interviewer: Dirk Luijk

1. Over Qualogy Suriname

Wat voor klanten heeft Qualogy en waar?
Zowel bedrijven als overheid. Niet alleen in Suriname, maar ook in Caribisch gebied.

Hoeveel mensen zijn er op dit moment in dienst? / Welke afdelingen zijn er?
Er zijn twee ontwikkelafdelingen, namelijk de Java-afdeling en de Oracle/Apex-afdeling. Op beide ontwikkelafdelingen werken ongeveer 10-12 mensen. Verder is er een testafdeling, waar drie testers en één testmanager werkzaam is. Tenslotte zijn er nog ondersteunende afdelingen zoals de afdeling Office Management en de directie.

In totaal zijn er bij Qualogy Suriname 30-35 mensen werkzaam.

Hoeveel projecten lopen er tegelijk?
Op dit moment lopen er 10 tot 15 projecten. Grote projecten die lopen zijn voor de Kamer van Koophandel (KvK) en de UTS. Elk project krijgt één, en soms twee testers toegewezen.

2. Het testproces

Werken jullie volgens TMap NEXT, en zo ja in hoeverre volgen jullie deze methode?
Ja, we werken wel volgens TMap Next, maar op een flexibele manier.

Wie stelt het functioneel ontwerp en het technisch ontwerp op?
Het functioneel ontwerp wordt meestal opgesteld door de ontwikkelafdeling, soms door de testafdeling. Requirements zijn inbegrepen in het functioneel ontwerp. Het technisch ontwerp wordt opgesteld door het ontwikkelafdeling.

Welke ontwikkelmethodes worden het meest toegepast?
Bij de Oracle-afdeling werken ze voornamelijk volgens de watervalmethode. Bij Java wordt er steeds meer met Scrum gewerkt.

Wat voor testsoorten en testvormen passen jullie zoal toe?
Functionele testen, load/stress testen en regressietesten.

Hoe zit het met unittesten? In hoeverre wordt dit toegepast?
Unittesten wordt nog nauwelijks, in ieder geval te weinig, toegepast. Op de testafdeling is het enigszins onduidelijk wat hierin de stand van zaken is. Maar over het algemeen wordt de software te gemakkelijk bij de testers afgeleverd.

Welke tools gebruiken jullie op dit moment bij het testen, en wat doen deze tools?
We gebruiken MindMap voor het maken van een testplan en Rapid Reporter voor het rapporteren van issues.

Is er een aparte afdeling/persoon die de testinfrastructuur inricht en beheert?
De afdeling beheer (drie mensen) zorgt voor de benodigde hardware/resources, en de developers richten tenslotte de omgeving in en installeren de te testen applicatie.

Wat zijn op dit moment de belangrijkste knelpunten?
De unittesten moeten gewoon beter, omdat deze ons veel werk kan besparen. Verder zijn we heel veel tijd kwijt aan het testen van (de validatie van) formulieren. Tenslotte moeten met name de regressietesten geautomatiseerd worden, omdat dit ons bij projecten met wijzigingen veel tijd kan besparen.

Hoe zit het met de acceptatietesten? Hoe vinden die plaats?
De acceptatietesten worden uitgevoerd door de klant. De wijze waarop dit wordt uitgevoerd, verschilt dus per project, maar vereist meestal geen voorbereiding en/of begeleiding vanuit Qualogy.

Hoe worden risico's in kaart gebracht? Is er sprake van een productrisicoanalyse?
Nee, dit gebruiken we niet. Risico's bepalen we eigenlijk voornamelijk zelf, maar dit wordt niet expliciet in kaart gebracht en vastgelegd.

Hoe vindt de go/no-go beslissing plaats voor het in productie nemen van een product?
Wanneer een product volgens de afgesproken kwaliteit wordt opgeleverd, wordt deze vrijwel altijd geaccepteerd door de klant.

Interview testafdeling

Vrijdag 5 september 2014

Inleiding

Op vrijdag 5 september is er een kort interview geweest met Ray Quartier en Gail Djaspan van de testafdeling bij Qualogy Suriname. Het interview ging over kwaliteitsverschillen tussen Oracle-projecten en Java-projecten.

Aanwezig: Ray Quartier, Gail Djaspan
Interviewer: Dirk Luijk

Dirk: Zijn er kwaliteitsverschillen tussen Oracle-projecten en Java-projecten?
Ray: Jazeker, ik vind dat Oracle-projecten over het algemeen van betere kwaliteit zijn.

Dirk: Wat bedoel je precies hiermee, zijn er bijvoorbeeld minder bugs bij de Oracle-projecten?
Ray: Nee, dat niet zo zeer. Er is een goede communicatie, en ook heeft de Oracle-afdeling een goede lead developer.

- Documentatie van Oracle-afdeling beter
- Qafé is niet altijd even stabiel
- Verschilt erg per project en per team
- Oracle-afdeling mogelijk iets volwassener, omdat Java-projecten nog wel eens door beginnende ontwikkelaars worden uitgevoerd

QUALOGY

Beschrijving uitgangssituatie

v.1.0

Afstudeeropdracht bij Qualogy Suriname

Door Dirk Luijk • 5 september 2014

Inhoudsopgave

Inleiding

In dit document wordt de bestaande situatie beschreven binnen de ontwikkel- en testafdeling bij Qualogy Suriname. Deze beschrijving vormt het uitgangspunt voor het onderzoek naar testautomatisering en kwaliteitsbewaking, dat wordt uitgevoerd door Dirk Luijk in de periode september - december 2014.

Allereerst wordt er een beschrijving gegeven van Qualogy Suriname, daarna wordt er vanuit organisatieniveau ingezoomd op de voor het onderzoek relevante afdelingen. Tenslotte worden er een aantal problemen beschreven die de reden vormen voor het onderzoek.

1. Algemeen

In dit hoofdstuk volgt een korte beschrijving van Qualogy.

Over Qualogy

Qualogy is een Nederlands bedrijf en is gevestigd in Rijswijk. Het is opgericht in 1998. Qualogy 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

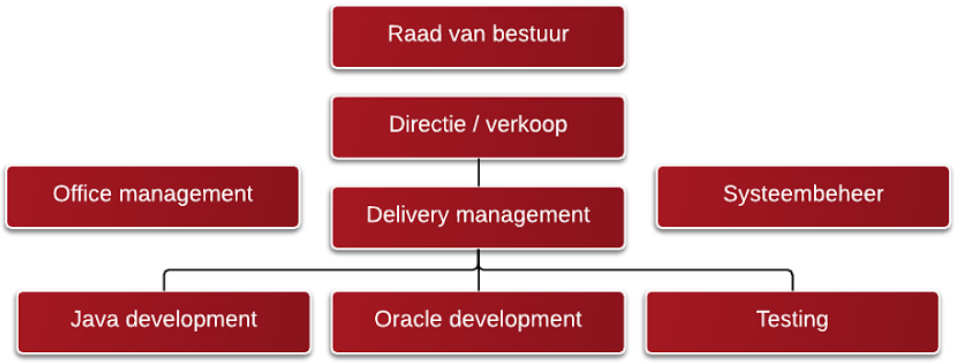
Qualogy Suriname is in 2007 opgericht en is gevestigd te Paramaribo. Inmiddels opereert Qualogy Suriname zelfstandig en heeft haar eigen klantenkring in Suriname en Caribisch gebied. Een sterk punt is dat Qualogy Suriname ook eigen personeel opleidt tot young professionals. Er zijn op dit moment ongeveer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

Organisatiestructuur

Allereerst is de structuur van de organisatie in kaart gebracht. Bij Qualogy Suriname is er sprake van een **lijn-staforganisatie**, maar zijn er ook kenmerken van een **matrixorganisatie**:

- Personen die hetzelfde werk doen, behoren tot dezelfde afdeling. Zo is er een testafdeling en een aparte ontwikkelafdeling 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. Een aantal projectgroepen zitten in een aparte ruimte, waar ook de stand-up meetings gehouden worden.

De organisatiestructuur is zichtbaar in onderstaand organogram.



2. Afdelingen

Binnen Qualogy Suriname is er veel diversiteit. Om te beginnen wordt er ontwikkeld in zowel Java als in Oracle/APEX. Terwijl de Java-afdeling bestaat uit vier teams (projectgroepen) en werken volgens Scrum, is de Oracle-afdeling meer één geheel en wordt er volgens een watervalmethode gewerkt. De testafdeling bestaat uit vier personen en maken deel uit van zowel Java- als Oracle-projecten. Sinds kort zitten de testers ook in dezelfde fysieke ruimte als de ontwikkelaars.

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. Daarom is men overgestapt op Rapid Software Testing, voornamelijk om veel papierwerk te voorkomen. 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 worden door de klant uitgevoerd.

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.

Java

De afdeling Java bestaat uit vier teams. Op het moment van schrijven zijn deze teams gekoppeld aan de volgende projecten:

Project	**Bezetting**	**Techniek**	**Werkzaam**
Telesur	3 ontwikkelaars 1 tester (parttime)	Java/Qafe	Op locatie
UCC	2 ontwikkelaars 1 tester (parttime)	Java/Grails	Op locatie
UTS	6 ontwikkelaars 1 tester (fulltime)	Java/Qafe	Zowel op kantoor als op locatie
KvK / GLIS	6 ontwikkelaars 1 tester (parttime)	Java/Qafe/Grails	Zowel op kantoor als op locatie

Het kennisniveau en de mate van kwaliteitsmanagement verschilt onderling. Zo past het UTS team al volledige *continuous integration* toe m.b.v. Jenkins, terwijl andere teams dit nog niet hebben. Ook de mate van unittesten verschilt onderling, al zijn harde cijfers m.b.t. de dekkingsgraad niet bekend.

De Java-teams werken allen volgens Scrum met sprints van 1 of 2 weken en gebruiken JIRA Agile voor het beheren van de scrum boards en de user stories.

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.

Sommige teams werken al met TDD, al is er niets bekend over de dekkingsgraad. Men is niet bekend met BDD.

Oracle

De Oracle-afdeling werkt volgens een watervalmethode. Men werkt in kleine groepen (2 á 3 man) aan projecten of soms zelfstandig. Er worden geen unittesten gebruikt, hoewel er bij een gedeelte van het team wel al draagvlak voor is.

Versiebeheer wordt niet toegepast, laat staan *continuous integration*. Sullivan (Oracle developer) denkt echter dat dit niet nodig is, aangezien de projectgroepen klein zijn en er vrijwel nooit conflicten zijn. Men werkt gezamenlijk in dezelfde ontwikkelomgeving.

3. Bevindingen

De volgende bevindingen zijn tijdens het onderzoeken van de bestaande situatie aan het licht gekomen.

Quality Control

Het belangrijkste probleem op de testafdeling is dat het handmatig uitvoeren van de tests veel tijd in beslag neemt. Met name bij het project 'KvK - Frontoffice' kost het heel veel tijd om steeds alle formulieren handmatig in te vullen.

Oracle

Bij Oracle wordt er alleen handmatig getest (error guessing). Enige vorm van unit testing wordt niet gebruikt. De meningen over het belang van unittesten zijn verdeeld.

Verder zijn de gerapporteerde issues (in JIRA) soms onduidelijk. Het zou mooi zijn als deze beter omschreven worden, mogelijk in een standaard format.

Tenslotte mag er meer gelet worden op de prioritering van testen. Als voorbeeld is genoemd het maximaal aantal karakters wat ingevuld mag worden. Op de afdeling heerst soms het gevoel dat er teveel aandacht besteed wordt aan triviale zaken die weinig risico's hebben voor de klant.

Java

Bij Java wil men graag meer aandacht besteden aan unittesten, maar men heeft er vaak te weinig tijd voor. Verder zijn de teams erg zelfstandig maar lijkt het kennisniveau te verschillen met team.

Concept adviesvoorstel

Afstudeeropdracht Qualogy Suriname

10 september 2014

1. Testafdeling

Op de testafdeling is gekeken in hoeverre de functionele (regressie)testen geautomatiseerd konden worden.

Selenium

De meest geschikte tool om mee te werken is tot nu toe Selenium. We hebben daar de afgelopen anderhalve week mee gewerkt bij het KvK-project en hebben resultaat geboekt. Zowel Kim, Gail als Ray Q. kunnen er nu mee werken. Bijna alle flows van de frontoffice zijn inmiddels geautomatiseerd, wat betekent dat bij een volgende sprint het doorlopen van de testgevallen een stuk sneller zal verlopen. Een inschatting van tijdbesparing volgt spoedig, op basis van cijfers welke Ray Quartier zal aanleveren.

Problemen

Voor bepaalde projecten zal het gebruik van Selenium relevant zijn, met name voor de regressietesten. Voor andere projecten zal het echter minder bruikbaar zijn. De volgende problemen worden namelijk nog niet opgelost met Selenium:

- ➔ De testdata zit opgeslagen in het testscript. Eenzelfde test meerdere keren draaien met verschillende data is daardoor niet mogelijk.
- ➔ Hergebruiken van stukken testscript is niet mogelijk. Wanneer een formulier voorkomt in meerdere testgevallen, moet dit dus op meerdere plaatsen aangepast worden.
- ➔ Wanneer de gebruikersinterface verandert, moet het testscript aangepast worden.

Hierbij ben ik er vanuit gegaan dat het huidige testteam geen uitgebreide programmeer-vaardigheden heeft en dat het te sten dit ook niet mag vereisen.

Mogelijke oplossingen

Op dit moment zijn er meerdere strategieën denkbaar die genoemde problemen kunnen oplossen.

1. De hoeveelheid onderhoud aan de testscripts accepteren en Selenium blijven gebruiken. Wanneer er weinig verandert aan de te testen onderdelen (wat bij regressietesten het geval is), zal er weinig onderhoud nodig zijn omdat de gebruikersinterface niet verandert. Mogelijk moet alleen de testdata per sprint geactualiseerd worden.
2. Op zoek gaan naar een uitgebreidere tool die bovenstaande problemen oplost door middel van een meer modulaire opbouw van de testscripts, maar waar geen programmeren aan te pas komt. Dit vraagt echter om behoorlijk uitgebreide software met een flink prijskaartje, voor zover nu bekend.
3. Test automation experts opleiden die programmeervaardigheden hebben en in staat zijn om de door Selenium gegenereerde code te optimaliseren in een JUnit omgeving. Testdata zou door het script uitgelezen kunnen worden uit een Excel of uit een database.

2. Ontwikkelafdelingen

Op de ontwikkelafdelingen is gekeken hoe het gesteld is met het kwaliteitsmanagement van de software.

Oracle-afdeling

Op de Oracle-afdeling werkt men in kleine teams van 1 - 3 man. Men werkt daarom tegelijk in dezelfde ontwikkelomgeving zonder dat dit problemen oplevert. Versiebeheer wordt niet gebruikt. Volgens Sullivan is dit geen probleem, vanwege de kleine teams.

Unittesten wordt nog niet gedaan op de Oracle-afdeling. De meeste ontwikkelaars zien het nut er niet van in en/of zijn niet gemotiveerd genoeg om het te gebruiken.

Java-afdeling

Onder de projectteams van Java is er veel diversiteitt. De meesten werken wel met versiebeheer, en unittesten wordt al gebruikt. Echter is het nog niet inzichtelijk hoe hoog de dekkingsgraad van de unittesten is.

Verder is er één Java-team (UTS) dat volledige *continuous integration* toepast door middel van Jenkins, inclusief deployment.

Voorlopig advies

Het advies voor de ontwikkelafdelingen is als volgt:

1. **Unittesten introduceren op de Oracle-afdeling**
   In samenwerking met Sullivan wordt unittesten ingevoerd op de Oracle-afdeling. Om het benodigde draagvlak te creëren, wordt er eerst een presentatie gegeven om de noodzaak en de voordelen van unittesten duidelijk te maken. Daarna worden er richtlijnen vastgesteld, zoals dekkingsgraad.
   Het invoeren van *continuous integration* (versiebeheer, automatische build en test, deployment) wordt niet geadviseerd, omdat dit teveel in één keer is, teveel tijd zou kosten en mogelijk ook niet zinvol is in de huidige situatie (kleine teams, watervalmethode).
2. **Continuous integration toepassen voor alle Java-teams**
   Aangezien de Java-teams bekend zijn met unittesting en gebruik maken van scrum in een groter team, is het tijd voor de volgende stap: volledige continuous integration d.m.v. Jenkins. De positieve ervaring van het UTS-team hiermee kan gebruikt worden om draagvlak te creëren bij de andere teams. De volgende stappen worden gezet:
   - Richtlijnen voor unittesten worden opgesteld (bijv. minimale dekkingsgraad).
   - Jenkins wordt ingericht voor alle teams (relevante informatie komt in het implementatierapport).
   - De teams krijgen één of meer workshops om met Jenkins te kunnen werken.

QUALOGY

TOOLSELECTIE

v.0.9.9

Tools voor test automation

Door Dirk Luijk • 30 oktober 2014

Inhoudsopgave

1. Inleiding

In dit rapport wordt de toolselectie beschreven die is uitgevoerd bij Qualogy Suriname. De toolselectie is uitgevoerd in het kader van een onderzoek naar de mogelijkheden op het gebied van test automation bij Qualogy Suriname.

1.1 Aanleiding

De toolselectie volgt op het voorlopige advies dat op woensdag 10 september 2014 is voorgelegd aan Qualogy Suriname. Er werd op dat moment al succesvol gebruik gemaakt van Selenium voor het automatiseren van functionele end-to-end testen. Selenium bleek echter, zo stond in het adviesrapport, ook enkele tekortkomingen te hebben. Een uitgebreider onderzoek naar alternatieven was daarom nodig.

Verder moet er ook geïnventariseerd worden welke tools ingezet kunnen worden op de ontwikkelafdelingen Java en Oracle/APEX, ter ondersteuning van unittesting en ter verbetering van de kwaliteitsbewaking.

1.2 Doelstelling

De doelstelling van deze toolselectie is om een shortlist op te stellen met zinvolle tools die binnen korte tijd ingezet kunnen worden bij Qualogy Suriname. Er wordt geselecteerd op tools die:

- helpen bij het automatiseren van unittesten op de ontwikkelafdelingen;
- ingezet kunnen worden in een omgeving met testers zonder programmeervaardigheden;
- leiden tot betere kwaliteitsbewaking van de geleverde software.

De algemene eisen worden beschreven in paragraaf 2.2.

1.3 Stakeholders

De volgende stakeholders zijn betrokken bij de toolselectie:

- **Management**
  Het management van Qualogy Suriname wil dat door middel van de tools de doorlooptijd van projecten verkort wordt, doordat er minder tijd aan testen besteed hoeft te worden. De tools mogen echter niet te veel geld kosten omdat het budget hiervoor beperkt is.
- **Testafdeling**
  De werknemers op de testafdeling moeten uiteindelijk met een tool kunnen werken die gebruiksvriendelijk is en weinig of geen programmeervaardigheden vereist.
- **Java-ontwikkelafdeling**
  De Java-ontwikkelafdeling bestaat uit meerdere teams en hebben belang bij tools die aansluiten op de huidige situatie en kennisniveau. Belangrijk is dat zij gemotiveerd zijn om meer testgedreven te gaan werken en hebben daarom belang bij een laagdrempelige en toegangelijke methode.
- **Oracle-ontwikkelafdeling**
  De Oracle-ontwikkelafdeling doet nog heel weinig met unittesten. Gezocht wordt naar eenvoudige tools die integreren in de bestaande ontwikkelomgevingen (IDE's).

1.4 Uitgangssituatie

De bestaande situatie is uiteraard bepalend voor de toolselectie. Deze wordt uitgebreid beschreven in het rapport 'Onderzoek bestaande situatie'.

Hieronder volgt alleen een overzicht van de reeds gebruikte tools.

De volgende (relevante) tools/software worden bedrijfsbreed gebruikt:

- **JIRA** - project management en rapporteren van issues

Op de testafdeling worden de volgende tools gebruikt:

- **MindMap**: voor het maken van een testplan
- **RapidReporter**: voor het bijhouden van testgevallen en bevindingen tijdens *exploratory testing*.

De volgende tools/software worden binnen de Java-teams gebruikt:

- **JIRA Agile** - beheren van scrum boards en user stories
- **Eclipse** - ontwikkelomgeving voor Java
- **Jenkins i.c.m. Maven** - het UTS-team maakt gebruik hiervan voor *continuous integration*

Verder ontwikkelt men bij Java in **Grails** of in **Qafé**.

De volgende tools/software worden bij de Oracle-afdeling gebruikt:

- **Oracle SQL Developer**: de gratis ontwikkelomgeving van Oracle
- **PL/SQL Developer**: een ontwikkelomgeving van Allround Automations
- **Toad:** een ontwikkelomgeving van Dell
- **APEX webomgeving**: voor het bouwen van de frontend

2. Werkwijze

In dit hoofdstuk wordt beschreven hoe de toolselectie is uitgevoerd. Allereerst wordt de gebruikte methode toegelicht, daarna volgen een aantal algemene eisen en tenslotte wordt uitgelegd welke indeling is gemaakt bij het onderzoeken van de tools.

2.1 Methode

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.

2.2 Algemene criteria

Bij het uitvoeren van de toolselectie wordt er, ongeacht de categorie, op de volgende zaken gelet:

- **mate van activiteit**, in hoeverre het product op dit moment nog in ontwikkeling is en wanneer de laatste versie is gepubliceerd;
- **volwassenheid**, hoe lang bestaat de software al, en zijn er handleidingen en boeken van;
- **integratie met andere tools**, is er bijv. een Eclipse plugin beschikbaar;
- **kosten / licentie**, de kosten moeten beperkt blijven; daarnaast heeft open-source de voorkeur;
- **compatibiliteit**, werkt de software in de bestaande omgeving;
- **leerbaarheid**, in hoeverre het aansluit op het bestaande kennisniveau.

2.3 Indeling tools

Op basis van hun kenmerken zijn de tools verdeeld in de volgende categorieën.

2.3.1 Web (unit)testing

Dit type testen kent vele uiteenlopende benamingen. Sommigen noemen het *functional testing*, anderen *web testing* of *web unit testing*. In feite zijn het geautomatiseerde end-to-end testscripts die uitgevoerd worden vanuit een xUnit omgeving. Hierbij wordt er een browser aangestuurd, dit kan een echte browser of een headless browser zijn.

Bij deze categorie wordt er gelet op integratie met de ontwikkelsoftware (IDE), syntax van de testcases en de mogelijkheden in combinatie met andere tools.

Op dit moment wordt er bij Java al gewerkt met JUnit. Bij de toolselectie wordt daarom o.a. gekeken of de webtests gebruikt kunnen worden in een JUnit omgeving.

2.3.2 BDD tools

Indien het zinvol is om Behaviour Driven Development in te voeren binnen Qualogy, moeten er ook tools zijn die dit mogelijk maken. De tools moeten scenario's kunnen inlezen en zijn in staat om op basis hiervan code 'stubs' te genereren, zodat de scenario's eenvoudig en snel omgezet kan worden naar testscripts.

2.3.3 Browser automation

Voor het automatiseren van end-to-end (web) testen, moet het gedrag van een browser gesimuleerd kunnen worden. Er zijn hiervoor twee oplossingen:

- **Headless browser emulators**: dit zijn browser emulators die volledig uitgevoerd kunnen worden zonder een GUI. Deze emulators zijn meestal goed op hoog niveau (HTTP requests), maar zeer beperkt op lager niveau (JavaScript, CSS). Het voordeel is echter dat deze emulators erg snel zijn en vanuit een console (command line) uitgevoerd kunnen worden.
- **In-browser emulators**: dit is een emulator die echte browsers kan aansturen. Het voordeel is dat alles ondersteund wordt (CSS, JavaScript, AJAX), echter heeft dit invloed op de snelheid.

Voor beide oplossingen wordt er naar een library gezocht. Ook wordt gekeken of er mogelijk een abstractielaag toegevoegd kan worden, zodat beide tools via dezelfde API aangestuurd kunnen worden.

2.3.4 Test frameworks

Verder wordt er gekeken of er gebruik gemaakt kan worden van een overkoepelend framework die diverse tools met elkaar kan integreren, en aanvullende functionaliteit aanbiedt zoals rapportage en integratie met een *continuous integration* tool.

2.3.5 Record & playback

Tenslotte wordt er ook gekeken naar een tool waarmee testscripts gemaakt kunnen worden met behulp van record en playback functionaliteit, zonder dat er geprogrammeerd hoeft te worden. Deze tools kunnen gebruikt worden op de testafdeling voor aanvullende systeemtests en kunnen dienen als aanvulling op de aanwezige unittests.

2.3.6 Overige tools

Verder wordt er gekeken naar tools voor:

- Code coverage (om de dekkingsgraad van unittests te meten)
- Code inspection (om de kwaliteit van de code te meten)

2.4 Beperkingen

Gezien de situatie gelden er de volgende beperkingen:

- Er vindt geen direct overleg plaats met softwareleveranciers. Alle informatie wordt uitsluitend ingewonnen op basis van informatie die via internet te vinden is, zoals de productwebsite, documentatie en reviews.
- Er wordt alleen geselecteerd uit tools waarvan er een gratis probeerversie beschikbaar is. Tools die bijvoorbeeld alleen op afspraak gedemonstreerd worden, zijn niet meegenomen in de toolselectie.

Verder wordt er voor continuous integration geen tool gezocht, omdat er op dit moment al één team met Jenkins werkt. Besloten is daarom om Jenkins als uitgangspunt te nemen.

3. Longlist

Dit hoofdstuk bevat de longlist van de toolselectie: de lijst met tools waaruit geselecteerd gaat worden. De lijst is opgebouwd volgens de indeling uit paragraaf 1.3 en is opgesteld aan de hand van de randvoorwaarden die hieronder worden genoemd.

3.1 Randvoorwaarden

Voor het opstellen van de longlist gelden er per categorie de volgende randvoorwaarden.

- Unit testing
  - Het framework moet integreren in de bestaande IDE's (zie paragraaf 1.4)
  - 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 (zie paragraaf 1.4)
  - 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).

3.2 Overzicht longlist

Web unit testing	**BDD tools**	**Browser automation**	**Test frameworks**	**Record & playback** & enterprise software
JWebUnit Helium Canoo WebTest Geb	Concordion Cucumber Easyb JBehave JDave Spock	*In-browser emulators* Selenium (WebDriver) *Headless browsers* HtmlUnit PhantomJS SlimerJS	Thucydides	Badboy iMacros Selenium IDE Sahi Oracle Application Testing Suite

Hieronder volgt er een vergelijking van de tools uit de longlist.

3.2.1 Web unit testing

In onderstaande tabel staan de web unit testing tools uit de longlist. De tools doen allemaal vrijwel hetzelfde: het aansturen van een browser met behulp van een testscript. De belangrijkste verschillen zijn de syntax en de browsers die ondersteund worden.

	Browser(s)	**Syntax**
JWebUnit	HtmlUnit Selenium/WebDriver (beta)	Java
Helium	Selenium/WebDriver	Java
Canoo WebTest	HtmlUnit	XML Groovy
Geb	Selenium/WebDriver	Groovy + jQuery selectors

Voor Oracle Apex-oplossingen kan er in principe gebruik gemaakt worden van dezelfde tools, alleen betekent dit dat er dan een Java-ontwikkelaar aan te pas komt.

3.2.2 BDD tools

BDD-tools kunnen specificaties ("Given-when-then", ook wel *Gherkin* genoemd door tools als Cucumber) omzetten naar code-stubs en vervolgens de tests uitvoeren. Bovendien bieden zij integratie met de ontwikkelomgeving (IDE) van de ontwikkelaar.

In de volgende tabel worden de BDD tools uit de longlist vergeleken.

	Syntax (specs/fixtures)	(IDE-)integratie	Volwassenheid	Voor- en nadelen
Concordion	Specs in HTML Fixtures in o.a. Java, Python, Fantom, Ruby	Eclipse (plugin) Integreert met JUnit	gestart in 2006 v1.4.4 (aug 2013) 1 developer	- Werkt met HTML in plaats van plain text - Geen Groovy ondersteuning
Cucumber	Specs in plain text Fixtures in 40+ talen, waaronder Java, Groovy, C#, Python, Ruby	command line Ant Maven IntelliJ (standaard) Eclipse (plugin) Jenkins	gestart in 2008 v0.1.6 (okt 2008) v1.3.17 (sep 2014) v2.0 beta (aug 2014) 6 developers	+ Uitgebreide rapportage, ook in Jenkins + Hoge volwassenheid Goede integratie
Easyb	Specs/fixtures in Groovy	command line Ant Maven IntelliJ (plugin) Eclipse (plugin)	gestart in 2008 v1.2 (2011) Niet meer actief sinds 2011	+ Eenvoudig te leren - Custom plugin nodig - Geen losse specs - Niet meer in ontwikkeling
JBehave	Specs in plain text Fixtures in Groovy, JRuby, Java	Ant Maven Eclipse (plugin) integreert met JUnit Hudson (Jenkins)	v2.0 (sep 2008) v3.9.5 (okt 2014) 6 developers	+ Genereert code stubs (templates) + Uitgebreide rapportage, ook in Jenkins + Hoge volwassenheid + Goede integratie - Configuratie is wat lastiger
JDave	Geen lose specs Fixtures in Java	Integreert met JUnit	v1.0 (2008) v1.3 (2011) Niet meer actief sinds 2012	- Niet meer in ontwikkeling - Alleen Java-ondersteuning
Spock	Geen lose specs Fixtures in Java, Groovy	Integreert met JUnit	Gestart in 2009 v0.1 (2009) v0.7 (2012) In ontwikkeling 1 developer	+ Flexibele syntax + Bevat mocking library - Geen tekstuele specs

3.2.3 Browser automation

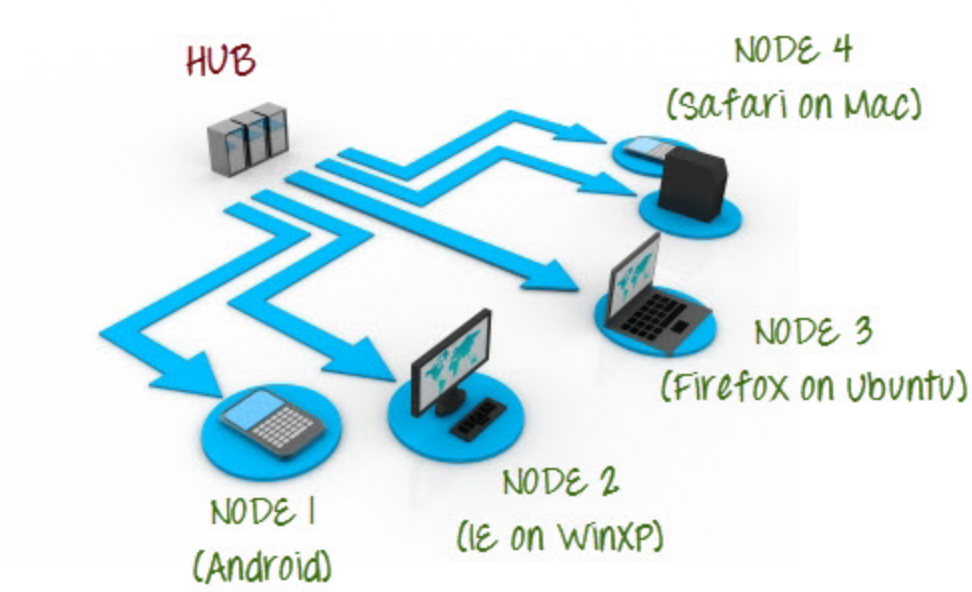
In-browser emulators: Selenium WebDriver
Voor het aansturen van 'echte' browsers (zoals Internet Explorer, Google Chrome) zijn is er slechts één tool die geschikt is voor de longlist: Selenium WebDriver (Selenium 2.0).

Mogelijke alternatieven zouden Sahi en Watir zijn. Sahi is vooral een record-en-playback tool en is geen direct alternatief voor Selenium WebDriver. Watir ondersteunde alleen Internet Explorer, maar maakt tegenwoordig ook gebruik van WebDriver.

De wat lastige syntax van Selenium kan 'vervangen' worden door gebruik te maken één van de tools uit de categorie 'web unit testing', zoals JWebUnit of Canoo WebTest.

Selenium Grid

Eén van de voordelen van Selenium is dat het ook een grid-configuratie ondersteunt, waarbij er op afstand parallel getest kan worden in meerdere browsers die WebDriver-ondersteuning bieden. Dit kunnen zowel echte browsers zijn (ook mobiele browsers) als headless browsers zoals HtmlUnit.



Headless browsers

Het nadeel van testen in echte browsers d.m.v. Selenium WebDriver is dat de tests erg lang kunnen duren. Het alternatief is om te testen met headless browsers. Dit zijn browsers zonder een GUI. De longlist bevat alleen headless browsers die ondersteuning bieden voor een configuratie met Selenium Grid (m.b.v. WebDriver implementatie) of direct in Java te gebruiken zijn.

	Engine	WebDriver support	Java-ondersteuning
HtmlUnit	Custom	Ja	Ja
PhantomJS	WebKit (Safari/Chrome)	Ja, met GhostDriver	Nee
SlimerJS (nog niet volledig headless)	Gecko (Firefox)	Ja, met GhostDriver* * momenteel nog in beta	Nee

De twee meest geschikte kandidaten lijken op basis van bovenstaande tabel HtmlUnit en PhantomJS te zijn. In de shortlist worden deze tools nauwkeuriger vergeleken.

3.2.4 Test frameworks

In de categorie *test frameworks* staat alleen Thucydides. Wat Thucydides doet, is voornamelijk het integreren van unit testing (JUnit) en BDD tools (JBehave, Easyb) met de testuitvoering en de rapportage (JIRA). Verder helpt Thucydides met het organiseren van web tests met de bijbehorende requirements en user stories, op de verschillende niveaus van de stakeholders.

Een nadere toolselectie voor deze categorie lijkt niet nodig te zijn, omdat er geen andere tools zijn gevonden met soortgelijke functionaliteit.

3.2.5 Record & playback

De volgende tools bevatten record en playback functionaliteit en kunnen gebruikt worden ter aanvulling op de reeds geautomatiseerde functionele unit- en browsertesten.

	Uitgever	**Platforms**	**Browsers**	**Kosten**
Badboy	Badboy Software	Windows	Internet Explorer	$45 eenmalig
iMacros	Ipswitch Inc.	Windows	Internet Explorer Chrome Firefox	gratis (browser add-on) $495 eenmalig (Standard) $995 eenmalig (Enterprise)
Selenium IDE	-	Windows Mac OS Linux	Firefox (IDE) + browsers met WebDriver	gratis
Sahi	Sahi			gratis (open source) $695 per gebruiker / jaar (pro)

Badboy
Badboy is een gratis tool voor Windows dat tests kan opnemen en afspelen op Internet Explorer. Het voordeel van Badboy is dat er met zowel HTTP-verkeer als met GUI-interactie gewerkt kan worden.

iMacros
iMacros bestaat uit een gratis versie in de vorm van browser addons, en kent verder twee betaalde versies Standard Edition ($ 495,00) en Enterprise Edition ($ 995,00). De betaalde versies ondersteunen ook Flash en Silverlight, bieden integratie met diverse platforms en ondersteunen koppelingen met databases.

Selenium IDE
De IDE-versie van Selenium is een browser add-on voor Firefox. Naast Firefox worden ook andere browsers ondersteunt wanneer men Selenium RC (WebDriver) installeert. Naast record-en-playback is Selenium IDE in staat om code te genereren voor diverse programmeertalen en -omgevingen, waaronder Java/JUnit 4.

Sahi
Het belangrijkste voordeel van Sahi is de brede ondersteuning voor browsers; verder claimt de ontwikkelaar dat het voornamelijk een gebruiksvriendelijke tool is. De syntax is in ieder geval eenvoudiger, bovendien worden er bij gebruik van AJAX geen 'explicit waits' in het testscript vereist, wat bij o.a. Selenium en BadBoy wel het geval is.

Sahi wordt overigens ook gebruikt als alternatief voor Selenium met behulp van de onderliggende API. Er wordt echter geen gebruik gemaakt van WebDriver, terwijl dit wel een W3C standaard wordt.

4. Shortlist

In dit hoofdstuk wordt de longlist door middel van een aantal selectiecriteria beperkt tot een shortlist van tools die ingezet kunnen gaan worden binnen Qualogy. Per categorie zullen de meest geschikte tools uitgelicht worden. In het hoofdstuk 5 zal er een framework worden voorgesteld op basis van de tools uit de shortlist.

4.1 Web unit testing

In deze paragraaf komen de tools aan bod voor web unit testing, ook wel *functional testing* of *browser testing* genoemd.

4.1.1 Selectiecriteria

De volgende selectiecriteria spelen een rol bij *web unit testing*:

- Ondersteuning voor zowel headless browsers als echte browsers;
- Eenvoudige syntax van de testscripts, zodat deze kort en begrijpelijk is;
- Integratie met Grails-omgeving, aangezien twee Java-teams hier inmiddels mee werken.

4.1.2 Vergelijking

Voor het automatiseren van de browsertests lijkt JWebUnit de meest geschikte tool vanwege ondersteuning voor zowel HtmlUnit als Selenium. Voor Selenium is er echter een plugin nodig die, volgens de ontwikkelaar, nog steeds in ontwikkeling is. De mate van volwassenheid van deze plugin is echter niet getest. De laatste versie van de plugin is van maart 2014.

Helium is vooral prettig vanwege de syntax, maar biedt alleen ondersteuning voor Selenium. Voor het testen op headless browsers is Selenium als tussenliggende laag dus wel vereist. Verder is Helium niet gratis maar kost € 149,00 per jaar.

Canoo WebTest biedt alleen integratie met HtmlUnit. Het heeft echter extra functionaliteit boven de tools die integreren met Selenium, zoals uitgebreide rapportage. Een minpuntje is dat de documentatie sterk uitgaat van XML als syntax voor de testscripts.

Geb heeft een eenvoudige syntax, vooral vanwege het gebruik van jQuery selectors. Als gratis tool lijkt deze dus geschikter dan Helium.

Tenslotte biedt Grails door middel van plugins alleen integratie voor HtmlUnit, Selenium WebDriver, Canoo WebTest en Geb. Een plugin voor Helium is er dus niet.

4.1.3 Conclusie

Op basis van genoemde afwegingen lijkt de meest flexibele oplossing om met JWebUnit te werken, zodat het gebruik van Selenium optioneel blijft. Mocht Selenium goed draaien, dan kan er overgestapt worden op Geb vanwege de gebruiksvriendelijke syntax, of op Canoo WebTest, indien de extra functionaliteit nodig blijkt te zijn.

4.2 BDD tools

In deze paragraaf komen de tools aan bod voor *Behaviour Driven Development*, ook wel *acceptance testing* genoemd.

4.2.1 Selectiecriteria

De volgende selectiecriteria spelen een rol bij BDD tools:

- Volwassenheid (aantal ontwikkelaars / grootte van de community)
- Specificaties in plain text heeft de voorkeur (zodat testers deze ook kunnen opstellen)
- Integratie met Thucydides

4.2.2 Afweging en conclusie

Bij de BDD-tools lijken Cucumber en JBehave de meest populaire tools te zijn, gelet op het aantal ontwikkelaars en de grootte van de community. Beide werken tevens met specificaties in plain text (Given-when-then, ook bekend onder de term *Gherkin*) en bieden goede integratie met diverse omgevingen.

Thucydides biedt echter alleen ondersteuning voor JBehave en Easyb. JBehave lijkt dus een logische keuze. JBehave is echter lastig te configureren en heeft een stijle leercurve. Daarom wordt geadviseerd om te beginnen met Spock als lichtgewicht tool.

Om ontwikkelaars te leren werken volgens BDD wordt in eerste instantie Spock geadviseerd. Later, al dan niet bij het in gebruik nemen van Thucydides, wordt er overgestapt op JBehave.

4.3 Browser automation

Voor browser automation kan er bij het gebruik van JWebUnit in eerste instantie zonder Selenium worden gewerkt, omdat deze ook directe aansturing van HtmlUnit ondersteunt. Echter wordt geadviseerd om op termijn met Selenium aan de slag te gaan om met echte browsers te kunnen testen. Bovendien kan er daarmee een *grid* worden opgezet, waarmee er tegelijkertijd op meerdere browsers kan worden getest.

Wat betreft de headless browsers kan er gewerkt worden met HtmlUnit, vanwege de goede WebDriver en JavaScript ondersteuning. PhantomJS is echter als veelbelovende nieuwkomer een geschikte aanvulling, omdat deze op WebKit draait en daardoor dicht in de buurt komt van Safari en Chrome. Het advies is daarom om deze naast elkaar te gebruiken wanneer er een *grid* wordt opgezet.

4.4 Record & playback

Voor record & playback is Selenium IDE de meest geschikte tool vanwege de integratie met Selenium RC (WebDriver) en Selenium Grid. Met andere tools is dit niet mogelijk. Bovendien kunnen de scripts eenvoudig geïntegreerd worden in een Java-omgeving. Zodoende kunnen de testscripts door middel van record-and-playback opgesteld worden door testers, waarna een developer deze scripts integreert in de JUnit-omgeving.

Als aanvulling op Selenium IDE is Badboy een zeer geschikte tool vanwege de hoge mate van gebruiksvriendelijkheid. Deze kan echter alleen op Windows met Internet Explorer testen, en stelt wel meer eisen aan de kwaliteit van de HTML-code.

Sahi bleek niet zo gebruiksvriendelijk te zijn als beloofd. Verder heeft deze een kleine open-source community, in tegenstelling tot Selenium.

4.5 Overzicht shortlist

Hieronder volgt een overzicht van de tools die overblijven in de shortlist.

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

5. Conclusie

Met behulp van de tools kan er gefaseerd een framework worden opgezet.

In de eerste fase van het framework wordt continuous integration ingevoerd. Hierbij wordt JUnit ingezet voor unittests, Maven als build tool, Jenkins voor het beheren van CI en Sonar voor het uitvoeren van code coverage en het inspecteren van de code quality. Browsertests kunnen dan nog semi-automatisch uitgevoerd worden door de testers met behulp van record & playback tools als Selenium IDE en Badboy.

Vervolgens kunnen browsertests worden geautomatiseerd. Dit kan met twee combinaties: JWebUnit met HtmlUnit (alleen headless browsers), of JWebUnit, Canoo WebTest of Geb in combinatie met Selenium Grid met HtmlUnit en PhantomJS (ook echte browsers).

Tenslotte kan *behaviour driven development* worden ingevoerd. Hiervoor kan Spock worden gebruikt, omdat deze zeer eenvoudig is en kan fungeren als opstapje naar BDD. Tenslotte overgestapt worden op JBehave in combinatie met Thucydides als framework, met JIRA integratie.

QUALOGY

ADVIESRAPPORT

v.0.9

Kwaliteitsverbetering door test automation

Door Dirk Luijk • 15 oktober 2014

QUALOGY

Samenvatting

In dit adviesrapport wordt er een advies gegeven voor het verbeteren van het ontwikkel- en testproces bij Qualogy Suriname, met als doel tijdsbesparing en kwaliteitsverbetering van de software. Door middel van interviews is de bestaande situatie bij Qualogy Suriname in kaart gebracht. Vervolgens is er met behulp van literatuuronderzoek gekeken naar best practices voor het optimaliseren van het testproces. Hieruit bleek dat *test automation* in combinatie met *continuous integration* de oplossing kan zijn en gebruikt kan worden om de genoemde doelstellingen te behalen. Een proefproject met Selenium heeft inderdaad bewezen dat *test automation* leidt tot tijdbesparing.

Automatiseren van testen begint al bij het ontwikkelen van software. Daarom moeten de ontwikkelaars meer aandacht besteden aan unittesten. Bij Oracle gebeurt dit nog niet en moet dit ingevoerd worden. Bij Java gebeurt het al, maar is er geen inzicht in de dekkingsgraad, dus moeten er tools worden ingezet om dit inzichtelijk te maken. Een dekkingsgraad van minimaal 80% moet verplicht worden voor zowel Oracle- als Java-projecten. Ook moet test-driven development gestimuleerd of zelfs verplicht worden, zodat unittests niet overgeslagen kunnen worden. Zowel het management als de ontwikkelaars moeten in het begin van elk project rekening houden met extra tijd voor het opstellen van unittests, zodat er later in het traject bespaard kan worden dankzij de reeds bestaande unittests.

Daarnaast kan het invoeren van *continuous integration* de ontwikkelcyclus versnellen, dankzij juist gebruik van versiebeheer, het automatiseren van het build-proces en het inzicht verkrijgen in de testresultaten. Voor Java-teams is de juiste kennis en tools aanwezig en wordt dit aangeraden. Voor de Oracle-afdeling wordt dit afgeraden vanwege het watervalkarakter van de projecten, de kleine teams, afwezigheid van versiebeheer en het ontbreken van geschikte tools.

Browsertests, ook wel end-to-end testen of acceptatietesten genoemd, kunnen bij Java ook worden geautomatiseerd. Quality Control kan gebruik maken van Selenium IDE voor het opnemen van afspelen van testscripts. Omdat Java-projecten een iteratief karakter hebben en het systeem vaak gewijzigd wordt, moeten de testers samenwerken met de ontwikkelaars voor het optimaliseren van de testscripts en het integreren ervan in de unittest-omgeving. In combinatie met *continuous integration* kan dit leiden tot een volledig geautomatiseerd testtraject waarbij er dagelijks automatisch in echte browsers getest wordt en de testresultaten direct zichtbaar zijn voor testers en manager.

Ook wordt er gekeken of requirements en testen op elkaar aangesloten kan worden. Dit is mogelijk met *behaviour driven development*. Optioneel wordt dit uiteindelijk ingevoerd binnen één of meer Java-teams.

De geadviseerde veranderingen zullen gefaseerd worden ingevoerd. Bij zowel Oracle als Java moet er na een bepaalde periode een bepaalde dekkingsgraad behaald zijn voor lopende projecten. Bij Java wordt eerst unittesting en continuous integration ingevoerd, later worden er automatische browsertests toegevoegd en optioneel wordt er tenslotte volgens *behaviour driven development* gewerkt. Genoemde implementaties zullen alleen bij het KvK-project plaatsvinden; bij andere projecten zal implementatie door iemand anders binnen Qualogy uitgevoerd moeten worden.

Tijdbesparing zal door middel van monitoring van urenregistratie en feedback van managers vastgesteld moeten worden. Kwaliteitsverbetering wordt inzichtelijk gemaakt dankzij *continuous integration* en het aantal JIRA-issues dat gerapporteerd wordt door Quality Control.

Inhoudsopgave

Inleiding

In dit adviesrapport wordt een advies uitgebracht voor het automatiseren en verbeteren van het testproces bij Qualogy Suriname. Aanleiding voor dit adviesrapport was de hoeveelheid tijd die de afdeling Quality Control nodig had om de regressietesten uit te voeren. Met name het project voor de Kamer van Koophandel van Aruba vereiste veel tijd en inspanning, omdat er voor de regressietesten veel formulieren steeds ingevuld moesten worden. Verder is Qualogy Suriname aan het overstappen op Agile ontwikkelmethodes zoals Scrum, waardoor het iteratieve karakter van projecten toeneemt en er vaker wordt getest.

Van augustus tot en met december 2014 loopt Dirk Luijk stage bij Qualogy. Zijn afstudeeropdracht bestaat uit het onderzoeken van de problemen die zich voordoen tijdens het testen en het adviseren en implementeren van een passende oplossing.

In dit adviesrapport wordt er advies gegeven welke veranderingen moeten worden doorgevoerd om het testproces te verbeteren en de hieronder geformuleerde problemen op te lossen.

Probleemstelling

De problemen die zich op dit moment voordoen, zijn:

1. Het testen van software gebeurt handmatig en kost daardoor veel tijd.
2. Er is geen inzicht in de interne kwaliteit van de software.

Het doel van de opdracht is dus tweeledig: er moet tijdsbesparing in het testproces gerealiseerd worden, en er is behoefte aan kwaliteitsverbetering van de geleverde software. Meer informatie over de probleemstelling is te vinden in het Plan van Aanpak (zie bijlage).

Opbouw van het rapport

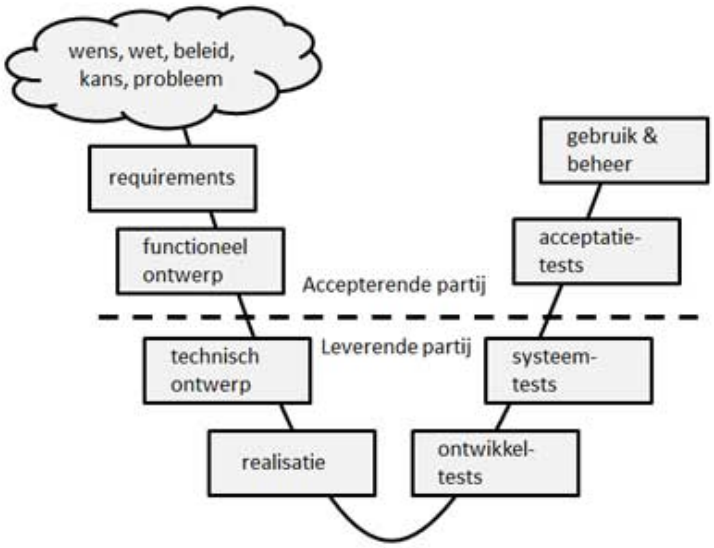
Het rapport begint met een stuk theorie waarin diverse zaken en begrippen worden uitgelegd. Vervolgens wordt in hoofdstuk 2 het uitgevoerde onderzoek beschreven en komt onder andere de toolselectie aan bod. Uit het onderzoek volgt een aantal mogelijke oplossingen, welke u kunt vinden in hoofdstuk 3. In hoofdstuk 4 wordt het rapport afgerond met het uiteindelijke advies.

1. Theorie

In dit hoofdstuk worden een aantal theoretische onderwerpen behandeld, waar later in het onderzoek en tijdens het bespreken van het advies naar gerefereerd wordt. Als basis voor de definities van begrippen is er gebruik gemaakt van TMap NEXT.

1.1 Het V-model

Om het gehele testproces in beeld te brengen wordt er vaak met het V-model gewerkt. Aangezien hiervan verschillende varianten bekend zijn, hanteren wij het V-model zoals deze beschreven wordt in TMap NEXT.



Figuur 1.1 Het V-model (bron: T-Map NEXT)

In figuur 1.1 vindt u aan de linkerkant de fasen waarin het systeem wordt gebouwd (of verbouwd), van wens tot realisatie. De diverse tussenproducten (de naamgeving kan in de praktijk verschillen) worden op elkaar getoetst. Aan de rechterkant van het V-model vindt het testen plaats. Hierbij worden de volgende testsoorten genoemd:

1. **Ontwikkeltests**: deze worden door de ontwikkelaars uitgevoerd. Hierbij vormen de technische specificaties de belangrijkste testbasis en is er inzicht in de werking van het systeem (white box). Hierbij wordt vaak op laag niveau getest, zoals de zogenaamde unittests en unit-integratietests (interactie met API).
2. **Systeemtests**: 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 (interactie met GUI).
3. **Acceptatietests**: deze worden uitgevoerd door de accepterende partij. Hierbij speelt vooral de verwachting een belangrijke rol.

1.2 Methodes voor test automation

In deze paragraaf worden een aantal methodes beschreven die gebruikt kunnen worden bij het automatiseren van een testproces.

1.2.1 Unittesten

De meest gebruikte methode om ontwikkeltests te automatiseren, is het gebruiken van *unittesten*. Bij unittesten schrijft de ontwikkelaar extra testscripts waarmee de normale code getest wordt. Hierbij worden afzonderlijke stukken code (modules, klassen) getest met verschillende scenario's (testcases).

Het voordeel van unittesten is dat men op een willekeurig moment in zeer korte tijd (binnen enkele minuten) kan vaststellen of de code nog naar behoren functioneert. De hoeveelheid code die gedekt wordt door unit tests, wordt ook wel de **dekkingsgraad** genoemd.

Aandachtspunten

Bij het gebruiken van unittesten gelden echter de volgende aandachtspunten:

- Wanneer alle unittests slagen, betekent dit niet dat het systeem geen fouten bevat. Dit hangt af van de dekkingsgraad enerzijds, en de juistheid van de reeds aanwezige unittests anderzijds.
- Een dekkingsgraad van 100% betekent niet per definitie dat het systeem klaar is om opgeleverd te worden. Het betekent alleen dat de aanwezige code zonder fouten door de unittests is gekomen. Echter kan het zijn dat er nog code met bijbehorende unittests ontbreekt.
- Unittests beperken zich tot de werking van code, de integratie van units wordt bijvoorbeeld niet getest. Ook wordt de externe kwaliteit van het systeem niet getest.

Integratie in ontwikkelomgeving

De meeste programmeertalen hebben wel een framework voor unittesting. Enkele voorbeelden zijn: SUnit (Smalltalk), JUnit (Java), NUnit (.NET), PHPUnit (PHP). Samen worden deze ook wel xUnit genoemd. Deze frameworks kunnen vaak ook geïntegreerd worden met de IDE, de ontwikkelsoftware van de programmeur, zodat er tijdens het programmeren met één druk op de knop unittests uitgevoerd kan worden, alvorens de code wordt opgeslagen in het versiebeheersysteem.

Test-Driven Development (TDD)

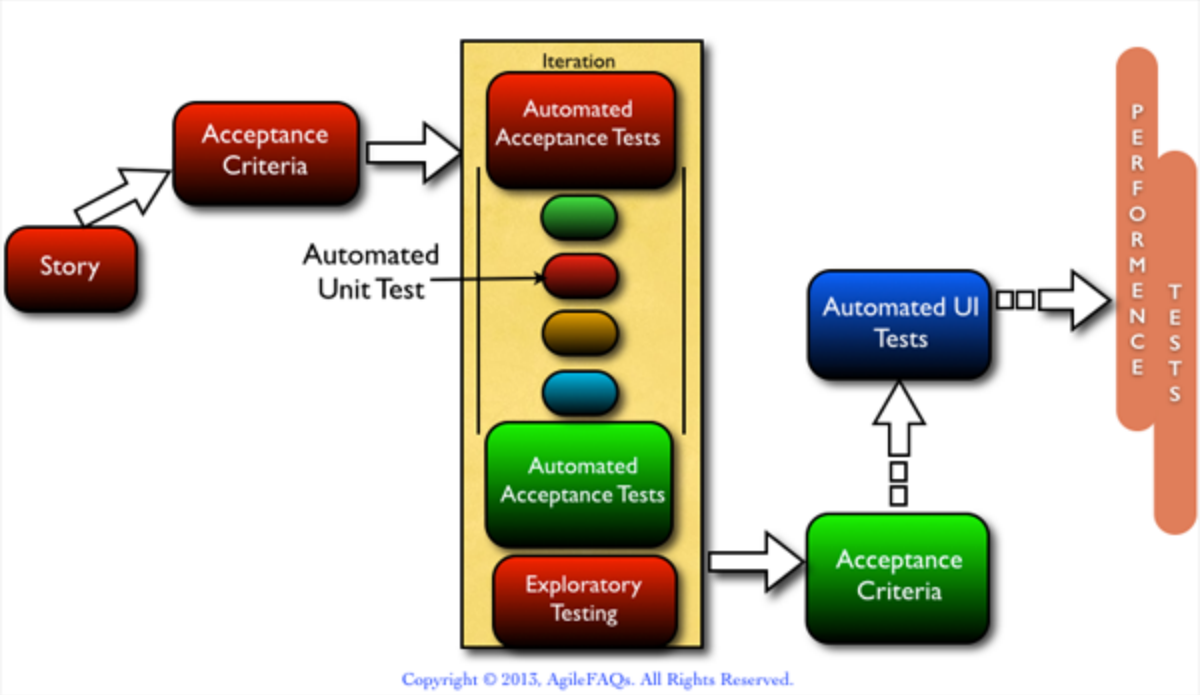
Normaliter worden unittests toegevoegd ná het schrijven van de code. Immers: testen volgt na realisatie. Het risico hiervan is echter dat de kwaliteit van de unittests lijden onder de tijdsdruk en dat daardoor niet alle code gedekt wordt door unittests.

Unittesten nemen echter een heel andere plaats in bij Test-Driven Development (TDD). Bij deze methode wordt bij het toevoegen of aanpassen van een functionaliteit éérst de automatische unittest geschreven, daarna volgt pas het schrijven van de code. Zodra de nieuwe unittest slaagt, is de functionaliteit correct geïmplementeerd en volgt het refactoren van de code.

Het voordeel van TDD is dat de unittests een leidende rol krijgen in het ontwikkelproces en niet overgeslagen worden. Verder stimuleert TDD een eenvoudige implementatie en een schone code. Een nadeel is dat de kwaliteit van de code hierdoor achteruit kan gaan; het refactoren mag daarom niet overgeslagen worden.

Behaviour Driven Development (BDD)

Een variant van Test-Driven Development is Behaviour-Driven Development (BDD). BDD is ontstaan doordat er bij TDD vaak te weinig aandacht was voor acceptatietesten. De unittesten werden opgesteld om te bevestigen dat het systeem doet wat het doet, en niet zozeer dat het doet wat de requirements voorschrijven.

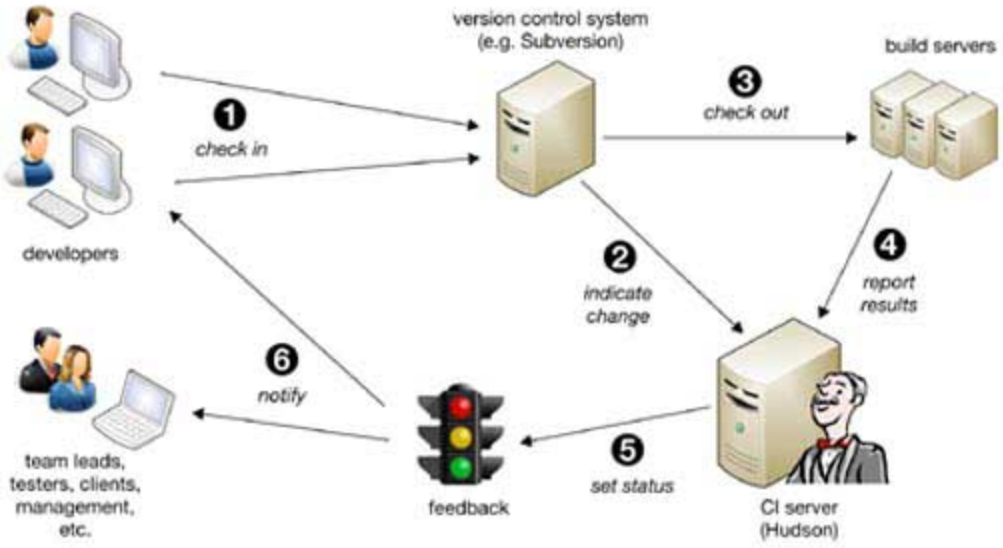


Figuur 1.2. BDD vs. TDD (bron: agilefaqs.com)

BDD is gebaseerd op dezelfde principes als TDD, echter spelen de acceptatietesten in vorm van *user stories* (uit de Agile ontwikkelmethodes) een belangrijke rol. De testcases moeten namelijk ook leesbaar zijn voor niet-programmeurs. Hierbij vormen de *scenario's* die voortkomen uit de user stories de basis. Het format van deze scenario's bevat meestal de notatie 'Given-When-Then' en wordt ook wel *specifications* of *Gherkin* genoemd.

1.2.2 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. De workflow van *continuous integration* is te zien in figuur 1.3.



Figuur 1.3. Workflow van continuous integration (bron: http://www.methodsandtools.com/)

Er zijn een aantal *best practices* bij bijdragen aan *continous integration*. Deze worden hieronder beschreven.

Versiebeheer

Voor het daadwerkelijk integreren van de code van de ontwikkelaars, wordt vaak een versiebeheersysteem (VCS) ingezet. Bekende open-source 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.

Automated deployment
Ook het uitrollen van de applicatie naar de test- of acceptatieomgeving kan geautomatiseerd worden. Een uitgebreide vorm hiervan is *Continuous Deployment*. Hierbij wordt de applicatie steeds na elke succesvolle build uitgerold naar de productieomgeving. Hiervoor moeten er echter vaak ook nog andere tests (zoals end-to-end tests) geautomatiseerd worden om regressie te voorkomen. Daarom wordt het systeem vaak alleen uitgerold naar de testomgeving.

Voor- en nadelen
Enkele voordelen van continuous integration zijn:

- Snelle feedback naar de ontwikkelaars toe over de kwaliteit, functionaliteit en de systeembrede impact van de code die ze ontwikkeld hebben;
- Constante beschikbaarheid van de huidige build voor test, demo of release doeleinden;

De nadelen van continuous integration zijn:

- Het kost tijd om het de eerste keer op te zetten;
- Er moeten voldoende unittests zijn om de voordelen te kunnen benutten;
- De hardware voor de build machines brengen extra kosten met zich mee.

1.2.3 Systeem- en acceptatietesten

Ontwikkeltesten zoals unittesten zijn vaak niet voldoende om een systeem volledig te testen. Het V-model (zie paragraaf 1.1) laat al zien dat er naast ontwikkeltesten ook systeem- en acceptatietesten zijn. Deze zijn nodig om het systeem op een ander niveau en met een andere testbasis te testen.

Een aantal van dergelijke testen zijn worden hieronder beschreven.

End-to-end testing
Bij end-to-end testen wordt een volledige flow tegelijk getest, in tegenstelling tot unittesten waarbij slechts in kleine gedeeltes wordt getest. Bij end-to-end tests is er vaak sprake van een interactie met de GUI of browser en wordt het gedrag van de gebruiker gesimuleerd. Het doel van end-to-end testing is om afhankelijkheden mee te testen; bovendien vormt het een extra garantie dat een applicatie werkt. Daarnaast is het een black-box methode die gericht is op de requirements van de opdrachtgever.

Voor het automatiseren van het end-to-end testen van webapplicaties kan er gebruik gemaakt worden van *browser automation*: software die het gedrag van een browser kan aansturen. Een bekend voorbeeld van een dergelijke tool is Selenium.

Het lastige van het automatiseren van dergelijke testen is dat het meestal technische kennis vereist van de testers, terwijl systeemtesten (en soms ook acceptatietesten) door testers worden uitgevoerd die over weinig of geen programmeervaardigheden beschikken. Voor het invoeren van end-to-end testing moet er daarom ook een programmeur in het testteam aanwezig zijn die de testscripts kan afwerken en eventueel kan integreren in de bestaande unittesten.

Niet-functionele testen

Ook sommige niet-functionele testen komen tijdens de systeem- en acceptatietesten aan bod. Voorbeelden hiervan zijn load- en stresstesten (performance) en tests met betrekking tot usability. Verder moet soms ook de layout van een applicatie (al dan niet in combinatie met diverse webbrowsers) worden getest. Dit is moeilijk te automatiseren, maar hoeft meestal ook minder vaak uitgevoerd te worden omdat er bij niet-functionele testen weinig of geen kans op regressie is na functionele wijzigingen.

2. Onderzoek

In dit hoofdstuk wordt het onderzoek beschreven dat is uitgevoerd binnen Qualogy Suriname.

2.1 Deelproblemen

De algemene probleemstelling is al beschreven in de inleiding van dit rapport. Hieronder worden een aantal deelproblemen beschreven die in dit onderzoek centraal stonden.

1. Welke tools kunnen binnen de bestaande testafdeling ingezet worden voor het automatiseren van de functionele (end-to-end) testen?
2. Welke tools kunnen binnen de bestaande ontwikkelafdelingen ingezet worden voor het automatiseren van de ontwikkeltesten?
3. Hoe kan er meer kwaliteitsbewaking ingevoerd worden op de ontwikkelafdelingen en hoe wordt de werkwijze hierop aangepast?

2.2 Methode

Tijdens het onderzoek zijn er diverse methodes gebruikt om de informatie te verzamelen en tot een passend advies te kunnen komen. Deze worden hieronder besproken.

Interviews

Er zijn diverse interviews gehouden om de bestaande situatie in kaart te brengen.

- Op maandag 18 augustus is er een interview geweest met Kim Ilahi en Gail Djaspan van Quality Control (testafdeling).
- Op woensdag 3 september is er een interview geweest met Sullivan Kromosoeto van de Oracle/Apex-afdeling.
- Op donderdag 4 september is er een interview geweest met Jozua Herfst van het Java-ontwikkelteam van project KvK.
- Op maandag 8 september is er een interview geweest met Rogér Pique en Ray Amatsoleman van het Java-ontwikkelteam van project UTS.

Middels deze interviews is geïnventariseerd hoe de bestaande situatie eruit ziet, met welke tools/methodes wel en niet gebruikt worden en wat de knelpunten zijn.

Literatuuronderzoek

Er is gebruik gemaakt van vakliteratuur en diverse (internet)artikelen voor theoretische achtergronden en ervaringen van professionals.

Informatie vanuit Qualogy Nederland

Op 8 oktober is er contact geweest met Erik Kerkhoven van Qualogy Nederland. Hij heeft adviezen gegeven voor het werken met unit testing en continuous integration. Verder heeft hij een aantal tools genoemd die ingezet kunnen worden voor het automatiseren van end-to-end testen.

Ervaringen van collega's

Verder is er tijdens pauzes en andere contactmomenten overleg geweest met collega's van zowel test- als ontwikkelafdelingen waarbij nieuwe informatie naar boven is gekomen.

Proefdraaien met Selenium
Voorafgaand aan de daadwerkelijke toolselectie is er al proefgedraaid met Selenium. Dit gebeurde in eerste instantie omdat er snel een oplossing nodig was voor het KvK-project, bovendien is Selenium een gratis tool dat zich prima leent voor dit doel.

Het proefdraaien met Selenium zorgde tevens voor nieuwe inzichten bij het voorbereiden van de toolselectie.

Toolselectie
De toolselectie draaide om het kiezen van een aantal tools die ingezet moeten gaan worden binnen Qualogy Suriname. Voor de volgende zaken is geprobeerd een tool te kiezen:

1. Een tool/framework voor **web unit testing**;
2. Een tool/framework voor **BDD** (Behaviour Driven Development);
3. Een geschikte basis voor **browser automation**;
4. Een compleet **test framework** die bovenstaande onderdelen met elkaar integreert;
5. Een **record-en-playback** tool voor de end-to-end (browser) testen binnen de testafdeling.

Voor categorie 1 t/m 4 is alleen gekeken voor tools die ingezet kunnen worden binnen de Java-teams, omdat er voor Oracle alleen unittesting ingevoerd gaat worden, al dan niet in combinatie met een record-en-playback tool voor browsertesten.

2.3 Resultaten

In deze paragraaf worden de resultaten van het onderzoek per beschreven methodiek uitgewerkt.

Interviews
De interviews zijn voornamelijk in gebruikt om de bestaande situatie in kaart te brengen. Verder is duidelijk geworden dat er veel scheiding is tussen ontwikkel- en testafdeling, wat de communicatie soms moeilijker maakt. Ook is de noodzaak van automatisering van het testproces en verbetering van unittesting duidelijker geworden.

> *De resultaten van het interview zijn uitgewerkt in het document 'Beschrijving bestaande situatie'.*

Proefdraaien met Selenium IDE
Aan de hand van de proefdraaien met Selenium is duidelijk geworden welke rol *record & playback* tools kunnen innemen in het geautomatiseerde testproces.

Voor bepaalde projecten zal het gebruik van Selenium relevant zijn, met name voor de regressietesten. Voor andere projecten zal het echter minder bruikbaar zijn. De volgende problemen worden namelijk nog niet opgelost met Selenium:

- Testdata zit opgeslagen in het testscript. Eenzelfde test meerdere keren draaien met verschillende data is daardoor niet mogelijk.
- Hergebruiken van stukken testscript is niet mogelijk. Wanneer een formulier voorkomt in meerdere testgevallen, moet dit dus op meerdere plaatsen aangepast worden.
- Wanneer de gebruikersinterface verandert, moet het testscript aangepast worden.

Selenium IDE is dus wel geschikt om snel een testscript te kunnen maken (prototyping), maar een programmeur zal uiteindelijk ook nodig zijn voor het optimaliseren van de testscripts en het integreren ervan in de unittest-omgeving, zodat de browsertests ook kunnen profiteren van de voordelen van continuous integration.

Toolselectie

Aan de hand van de toolselectie is er een toolselectierapport opgesteld, waarin een framework wordt voorgesteld dat gefaseerd ingevoerd kan worden op de Java-afdeling van Qualogy. Het framework omvat continuous integration in combinatie met geautomatiseerde web unittests en tools voor behaviour driven development.

> *Meer informatie over de toolselectie is te vinden in het het toolselectierapport.*

3. Advies

In dit hoofdstuk wordt er een advies gegeven op basis van het uitgevoerde onderzoek.

3.1 Overzicht

In deze paragraaf wordt een overzicht gegeven van de veranderingen die geadviseerd worden voor de verschillende afdelingen.

3.1.1 Oracle

Voor Oracle-projecten wordt alleen automatische unittesten geadviseerd. Voor het creëen van draagvlak wordt er een presentatie gegeven en een aantal workshops georganiseerd. Ook moet er een bepaalde dekkingsgraad verplicht zijn, 70% is minimaal, 80% is aanbevolen en 90% of hoger is optimaal.

Continuous integration is niet geschikt voor de Oracle-project, vanwege het gebruik van een watervalmethode en het ontbreken van een iteratieve methode zoals Scrum. Dit betekent dat browsertests nog steeds uitgevoerd moeten worden door testers. Dit kan gedeeltelijk geautomatiseerd worden met Selenium IDE.

3.1.2 Java

Voor Java-projecten wordt geadviseerd om continuous integration in te voeren, waarbij er door middel van code coverage en de resultaten van de unittests snel inzicht verkregen kan worden in de kwaliteit van het systeem. Ook worden de browsertests volledig geautomatiseerd door deze in de unittestomgeving te integreren.

3.1.3 Quality Control

Wanneer er tests geautomatiseerd gaan worden, zullen we werkzaamheden van de testers ook veranderen. Voor Oracle-projecten zullen de de browsertests nog altijd door de testers uitgevoerd moeten worden, maar voor Java-projecten zullen deze testen geautomatiseerd worden door het ontwikkelteam. Testers zullen meer moeten samenwerken met Java-ontwikkelaars voor het maken van testscripts en wanneer de tests geautomatiseerd zijn, zullen zij zich meer bezighouden met niet-functionele tests (usability, layout, vormgeving) en acceptatietesten.

3.2 Organisatie

In deze paragraaf wordt dieper in gegaan op het organisatorische aspect van het advies.

3.2.1 Java

Wat betreft de Java-projecten staat het testen nog teveel los van het ontwikkelproces. In het kader van Scrum is het belangrijk dat testers daadwerkelijk deel gaan uitmaken van het ontwikkelteam waarbij zij betrokken zijn. Het advies is daarom dat elk Scrum-team een eigen dedicated tester krijgt die ook in dezelfde ruimte gaat werken.

Omdat er op dit moment te weinig testers zijn om dit te kunnen realiseren, wordt geadviseerd om te kijken of er ook ontwikkelaars zijn die ook de rol van tester kunnen opnemen. Een team heeft namelijk geen fulltime tester nodig, bovendien is een ontwikkelaar ook in staat om zelf zijn testscripts op te stellen en te implementeren.

3.2.2 Oracle

Bij Oracle-projecten hoeft er weinig te veranderen. Het is belangrijk dat de testers die betrokken zijn bij de projecten in dezelfde ruimte blijven werken zodat er gemakkelijk gecommuniceerd kan blijven worden tussen ontwikkelaars en testers.

3.2.3 Management

Het management moet rekening houden met andere verhoudingen binnen de planning. Op dit moment wordt er in het begin van een project snel ontwikkelt, maar vindt er later in het project veel vertraging plaats vanwege de toegenomen complexiteit in combinatie met een tekort aan unit tests. Bij het invoeren van stricte unit testing en continuous integration zal er juist in het begin van het project veel tijd besteed worden aan unit testing, maar zal het project na verloop van tijd juist sneller verlopen dankzij minder bugs en meer vertrouwen en inzicht in de kwaliteit. Ook testers zullen veel moeten samenwerken met ontwikkelaars voor het maken van testscripts, maar zullen zij na verloop van tijd nauwelijks meer handmatig behoeven te testen en alleen nog enkele niet-functionele- en acceptatietesten uitvoeren.

3.4 Return on investment

Uit het proefproject met Selenium bij KvK is gebleken dat alleen het gebruik van Selenium IDE al leidt tot tijdbesparing. In deze paragraaf wordt hierin inzicht gegeven.

Handmatig testen:

- x formulieren, x testgevallen
- per testgeval x minuten
- totaal: x minuten per sprint

Automatisch testen:

- opstellen script voor formulier: x minuten (eenmalig)
- aanpassen van script voor andere testgevallen: x minuten (aanmalig)
- runnen/verbeteren van script: x minuten
- totaal: x minuten per sprint

3.4 Inzetten tools

Geadviseerd wordt om gefaseerd te gaan werken met de tools uit de toolselectie. De volgende fases worden voorgesteld.

Fase 1: unit testing (TDD) + continous integration

- Unit testing is verplicht (JUnit)
- Continuous integration wordt toegepast (Jenkins+Maven)
- Sonar wordt gebruikt om de dekkingsgraad vast te stellen en inzicht te geven in code quality
- Dekkingsgraad van minimaal 80% is verplicht

Fase 2: web unit testing (functional testing)

- Web unit testing wordt ingevoerd (JWebUnit, Canoo WebTest of Geb)
- Eerst wordt er alleen met JWebUnit en HtmlUnit gewerkt, later wordt er een Selenium Grid opgezet zodat er naast headless browsers ook op echte browsers kan worden getest. (virtual machines)

Fase 3: acceptance testing (BDD / ATDD)

- Requirements en unittesting worden in elkaar geïntegreerd d.m.v. *behaviour driven development*
- Eerst wordt er met Spock gewerkt, later wordt Thucydides en JBehave ingezet om ook integratie te bieden met o.a. JIRA voor automatische reports.

4. Implementatie

In dit hoofdstuk wordt een implementatieplan beschreven voor het invoeren van de geadviseerde veranderingen binnen Qualogy Suriname.

4.1 Scope implementatie

Bij Oracle wordt unittesting dusdanig ingevoerd, dat men zelfstandig unittests kan maken en dat de werkwijze van het team erop wordt aangepast.

Bij Java vindt de implementatie alleen plaats bij het KvK-project. Wanneer de implementatie daar is afgerond, kan de implementatie overgenomen worden binnen andere Java-teams. Er zal een implementatieplan worden opgeleverd om dit mogelijk te maken.

Verder zullen alleen fase 1 en 2 uit paragraaf 3.4 opgeleverd worden. Fase 3 (*behaviour driven development*) wordt alleen ingevoerd wanneer er tijd overblijft, anders zullen de instructies hiervoor meegenomen worden in het implementatieplan.

4.2 Richtlijnen

Bij zowel Oracle als bij Java wordt er voor een lopend project de dekkingsgraad langzaam opgehoogd:

- **35%** dekkingsgraad vóór **december 2014**
- **55%** dekkingsgraad vóór **februari 2015**
- **80%** dekkingsgraad vóór **maart 2015**

Met ingang van maart 2015 moet elk project een minimale dekkingsgraad hebben van 80%.

4.2 Activiteiten

Er zullen een aantal activiteiten georganiseerd worden voor het introduceren van nieuwe tools en concepten binnen Qualogy Suriname.

Event	**Onderwerp**	**Doelgroep**
Presentatie	Het belang van unittesten & TDD	Ontwikkelaars (Java en Oracle)
Demonstratie	Selenium IDE: Tips en trucs	Ontwikkelaars (Java en Oracle) Testers

4.3 Planning

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 17 okt - 28 nov	Web testing opzetten (Selenium)	- Werkende Selenium tests in KvK - 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)

Glossary

Requirements
Testen
Deployment
Continuous integration
Test automation
Unittests
Systeemtests
IDE
Unittests
JUnit
Commit
GUI