Professionele Bachelor Toegepaste Informatica

Optimalisatie geautomatiseerde testen en onderzoek naar Model Based-Testing

Boo Strackx

Promotoren:

Bram Thys — Refleqt
Wim Vervust — Refleqt
Luc Doumen — Hogeschool PXL

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Optimalisatie geautomatiseerde testen en onderzoek naar Model Based-Testing

Boo Strackx

Promotoren:

Bram Thys — Refleqt
Wim Vervust — Refleqt
Luc Doumen — Hogeschool PXL

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Het succesvol afronden van een stage had ik nooit kunnen doen zonder de hulp van verschillende personen. Hoewel ik een groot deel nog steeds zelf heb volbracht, zou ik nooit zo ver zijn geraakt zonder het deskundige advies van mijn bedrijfspromotor, Bram Thys. Daarnaast kreeg ik een extra duwtje in de rug van Wim Vervust, zonder hem zou ik waarschijnlijk nog steeds tegen een muur van fouten opkijken. Verder mag ik Floriaan Zandijk, mijn mede-stagiair, zeker niet vergeten. Hij heeft me meerdere keren met raad en daad bijgestaan. Intern hebben zij allemaal ervoor gezorgd dat mijn vragen nooit onbeantwoord bleven en dat eventuele obstakels zo snel mogelijk geruimd werden.

Langs de andere kant werd ik onder de vleugels genomen door Luc Doumen. Als hogeschoolpromotor zorgde hij ervoor dat ik me goed voelde bij het verloop van mijn stage en hoorde hij (regelmatig) mijn twijfels aan. Op bijna elk moment kon ik hem bereiken en hielp hij met het zoeken naar oplossingen voor problemen van welke aard ook. Meneer Doumen stond altijd voor me klaar en dit betekent nog steeds ontzettend veel voor mij. Ik kon me geen betere hogeschoolpromotor wensen en wil hem daar uitgebreid voor bedanken.

Binnen de PXL kon ik op nog iemand altijd rekenen, namelijk Nathalie Fuchs. Eigenlijk doet mevrouw Fuchs het werk van minstens drie personen, maar toch wordt dit naar mijn mening niet vaak genoeg erkend. Ook zij stond altijd klaar voor mij. Ze bood al mijn hele carrière op de hogeschool een luisterend oor, maar toch wou ik haar bijdrage even benadrukken. Ook al had ze eigenlijk geen tijd, toch vond ze ruimte in haar schema om mij of anderen vooruit te helpen. Er zijn veel mensen met een goed hart, maar weinigen blijven het van dag tot dag tonen. Mevrouw Fuchs is er een uit de duizenden.

Uiteindelijk vergeet ik zeker niet degenen die me hebben geholpen om zo ver te raken, namelijk vrienden en familie. Op de PXL heb ik verschillende mensen leren kennen. Elke persoon heeft me doen groeien, maar de personen die ik nu vrienden kan noemen, blijven me het meest bij. Door hen ben ik een fantastisch persoon geworden en heb ik dingen durven doen die ik mogelijk nooit had durven doen. Ze bleven me pushen en steunen zodat ik uiteindelijk nu hier sta, aan het einde van mijn schoolcarrière. In het bijzonder wil ik Niels Machiels bedanken, aangezien hij mij onzelfzuchtig advies en extra uitleg heeft gegeven over verschillende vakken. Het is door zijn geduld dat ik door mijn programmeervakken geworsteld ben.

Tot slot wil ik met het meest belangrijke afsluiten, namelijk mijn familie. Aangezien ik uit een niet zo vanzelfsprekende middelbare richting kom, namelijk Humane Wetenschappen, had niet alleen ik, maar ook anderen heel wat twijfels over mijn beslissing om Toegepaste Informatica te gaan doen. Degenen die altijd achter me stonden, zijn nog steeds mijn ouders en de rest van mijn familie. Ze zagen hoe graag ik hiervoor wilde gaan en ze zijn nooit gestopt met in mij te geloven. Ik ben altijd al een onzeker persoon geweest, dus het feit dat ze zoveel potentieel in mijn capaciteiten zagen, heeft ervoor gezorgd dat ik nooit heb opgegeven. Zonder zo een sterk kader om me heen van zowel vrienden als familie, had ik nooit gestaan waar ik nu sta.

Abstract

Het doel van deze stage is om enerzijds te leren hoe geautomatiseerde testen met behulp van Selenium en Cucumber opgezet kunnen worden en anderzijds om het fenomeen *Model Based-Testing* (MBT) te onderzoeken. Dit gebeurt door een eerder opgestart project genaamd de 'Crap App' verder af te werken en de eerder opgestelde geautomatiseerde testen beter op punt te stellen. Verder wordt de werking van MBT toegelicht aan de hand van een verbreding van algemene kennis over het onderwerp, een vergelijksmatrix op te stellen van een aantal tools die MBT toepassen en tot slot een conclusie te trekken.

Het opzetten van de geautomatiseerde testen wordt gerealiseerd door gebruik te maken van twee tools, namelijk Selenium en Cucumber. Selenium maakt het mogelijk om handelingen in een browser, zoals Google Chrome of Firefox, te automatiseren. Het aanklikken van een knop, het surfen naar een bepaalde website of het ingeven van een wachtwoord zijn allemaal browserhandelingen die dankzij Selenium door een script kunnen worden uitgevoerd. Zodra het script (correct) geschreven is, handelt de computer het verder af.

Cucumber is de technologie waarmee de testen opgesteld worden. Door gebruik te maken van Gherkin kan een tester in redelijk leesbare tekst de fundering van een test uitschrijven. Gherkin is de syntax of de grammatica waaraan een tester zich moet houden om een test te schrijven. Om uiteindelijk effectieve code te koppelen aan de leesbare tekst van de test, wordt de andere zijde van Cucumber gebruikt, het gedeelte dat ervoor zorgt dat code vastgehangen wordt aan de leesbare tekst van een test. Samen met Selenium wordt het mogelijk om geautomatiseerde testen te schrijven.

Het andere gedeelte van deze stage bestaat uit een onderzoek over *Model Based-Testing*. MBT is een techniek binnen testing dat op basis van modellen van (een gedeelte van) een programma testcases genereert. Als een tool dit kan doen voor een tester, bespaart dit de tester enorm veel tijd. Tijd is geld, dus kan het toepassen van een MBT-tool ook een voordeel opleveren voor het bedrijf zelf. Voor Refleqt zijn drie tools vergeleken: Conformiq Creator, MBTsuite en Yest. Bij het vergelijken van deze tools wordt ontdekt dat er veel meer komt kijken bij het principe van MBT. Zo bieden bepaalde tools de tester een aantal strategieën van testgeneratie aan zodat de tester de exacte testcases kan genereren die hij nodig heeft. Op basis van vereisten die Refleqt aangegeven heeft, wordt uiteindelijk één tool als winnaar bekroond.

Tot slot is Conformiq Creator de tool die het beste aansluit bij de noden van Refleqt. Niet alleen kan deze tool werken met diagrammen op basis van BPMN, een modeleringstandaard, ook aan CI-mogelijkheden en andere automatiseringsmogelijkheden is gedacht. Bovendien zijn er uitbreidingsmogelijkheden aan de hand van extra tools die door Conformiq worden aangeboden en is er voldoende documentatie ter beschikking voor een instapklaar gebruik.

Bij het testing gedeelte zijn twee zaken gerealiseerd. Allereerst zijn de originele testen van de Crap App opgeruimd en zijn ze nu makkelijker in gebruik, zeker naar de toekomst toe. De hoeveelheid code is geoptimaliseerd en verschillende functionaliteiten zijn eenvoudiger terug te vinden. Verder is een grote test opgesteld die alle functionaliteit van de home pagina test, met name het loginproces, het registratieproces en het gedeelte dat een vergeten wachtwoord of gebruikersnaam afhandelt. Naar de toekomst toe is er nog veel ruimte voor uitbreiding en is de Crap App een ideaal project om beginnende testers aan te zetten.

Inhoudsopgave

Lijst van gebruikte figuren

Lijst van gebruikte tabellen

Lijst van gebruikte afkortingen

ALM	*Application Lifecycle Management (tool)*
BDD	*Behavior Driven Development*
BDD	*Behavior Driven Development*
BPMN	*Business Process Modeling Notation*
CD	*Continuous Delivery*
CI	*Continuous Integration*
CRM	*Customer/Business relationship management*
FSM	*Finite state machine*
IT	*Information Technology*
MBT	*Model Based-Testing*
POM	*Page Object Model*
SDLC	*Software Development Life Cycle*
SUT	*System under test*
UML	*Unified Modeling Language*

Verklarende woordenlijst

Software Development Life Cycle	De SDLC is een proces voor het plannen, maken, testen en opstellen van een informatiesysteem. [35]
Version control system	Een versiebeheersysteem is een programma of een verzameling van programma's dat digitale informatie zoals computerbestanden kan beheren. [34]
Nice-to-have	Een eigenschap van een product dat fijn zou zijn om te hebben, maar dat mogelijk niet praktisch in uitvoering is tijdens de ontwikkelfase van een project. Het wordt meestal gezien als een 'leuk extraatje'. [33]
BPMN	*Business Process Modeling Notation*, een modeleringstandaard voor het opstellen van business modellen, bijvoorbeeld *activity diagrams*.

Inleiding

Elk eindwerk heeft zijn begin, hier is dat niet anders. Vandaar dat deze inleiding een globaal beeld schept van dit eindwerk en een gids vormt voor het begrijpen van dit document.

Dit werk bevat de opgedane ervaring in verband met enerzijds het opzetten van geautomatiseerde testen met behulp van twee testing tools, namelijk Selenium en Cucumber. Anderzijds wordt het onderzoek naar een opkomende trend in het testing milieu toegelicht, namelijk *Model Based-Testing* (MBT).

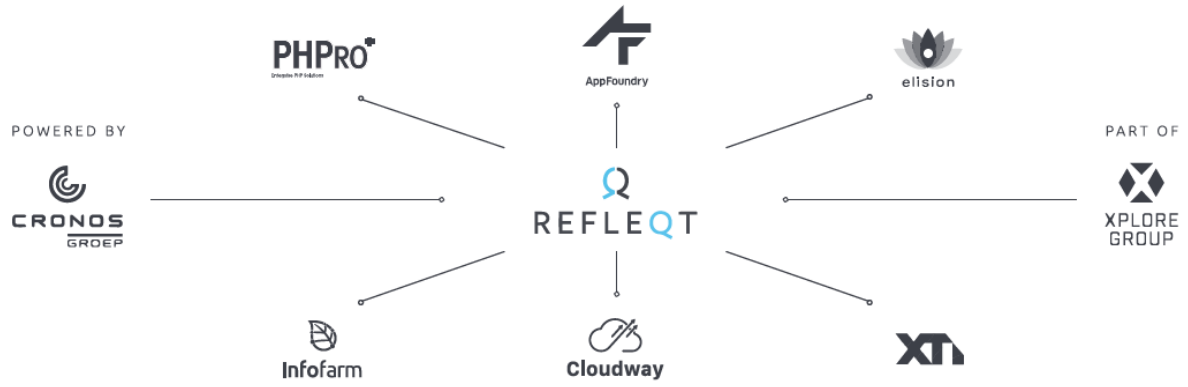
Het eerste gedeelte, de stageopdracht, beschrijft hoe een verbeterde versie van al bestaande geautomatiseerde testen wordt opgezet binnen de Crap App. De Crap App vormt het eindproduct van een afgelopen IT-project. Het is een trainingsapplicatie voor beginnende testers om hen te helpen met het vinden van bugs, dit zijn defecten die in software zitten. De applicatie is speciaal ontwikkeld voor dit doel en dus ideaal om een junior tester op los te laten. Op het eerste zicht lijkt de applicatie een normale app, maar er zit een addertje onder het gras. De app bevat schakelbare bugs. Een tester kan de bugs aan- of uitzetten en ze vervolgens proberen te vinden. Zo kan de tester allerlei bugs terugvinden en kijken hoe goed hij is in het vinden van deze bugs.

Vervolgens wordt in het tweede gedeelte van deze stage het principe van MBT onderzocht. Eerst wordt een beeld van MBT geschetst door de geschiedenis te beschrijven, de voor- en nadelen te bespreken en een heldere betekenis te definiëren. Hierna worden MBT-tools die gebruik maken van *Business Process Modeling Notation* (BPMN) met elkaar vergeleken op basis van eigenschappen die Refleqt, het stagebedrijf, belangrijk vindt, zoals bijvoorbeeld de mogelijkheid tot integratie in een automatische pipeline. Deze vergelijking gebeurt aan de hand van een vergelijkingsmatrix; een bepaalde score wordt gegeven aan verschillende onderdelen van elke tool op basis van de kritieke eigenschappen van Refleqt, om zo via een eindscore te bepalen welke tool de betere is voor Refleqt. Om het gedeelte af te sluiten, wordt een advies opgemaakt voor Refleqt en bijgevolg worden conclusies getrokken.

I. Stageverslag

1 Bedrijfsvoorstelling

Refleqt maakt deel uit van een groter bedrijf genaamd Xplore Group. Xplore Group is dan weer een tandwiel binnen een nog grotere machine, namelijk Cronos Groep. Binnen Xplore Group bevinden zich verschillende kleinere bedrijfjes met elk een specialiteit. De specialiteit van Refleqt is het waarborgen van kwaliteit.



Figuur 1: Refleqt binnen Xplore Group en Cronos Groep

Refleqt profileert zich in haar grondige kennis over de inzet van *test automation* voor *Continuous Delivery*. Daarnaast heerst er een open werksfeer waar ruimte is voor persoonlijke groei. Verder is Refleqt in staat een soort totaalpakket van kwaliteit aan te bieden, dit gaat van requirement analyse tot de training van eindgebruikers. Tot slot zet Refleqt in op de laatste nieuwe technologieën en richt het bedrijf zich vooral op complexere IT-omgevingen.

Binnen Refleqt wordt erg veel aandacht geschonken aan het uitwerken van een *continuous delivery* (CD) of *deployment* strategie voor elke klant. Concreet betekent dit dat de klant bijzonder nauw betrokken wordt vanaf het begin van een project. Refleqt staat tussen de klant en het development team in als een single point of contact. De klant kan in principe altijd terecht bij Refleqt voor de meest diverse vragen.

Om uitstekende kwaliteit te allen tijde te kunnen leveren aan een klant, focust Refleqt erg veel op automatisering. Dit bespaart niet alleen tijd en geld, ook (menselijke) fouten worden tot een absoluut minimum gehouden. Door zowel CD als automatisering te combineren, slaagt Refleqt erin op hoog tempo telkens nieuwe functionaliteiten op te leveren voor eindgebruikers.

2 Voorstelling stageopdracht

2.1 Probleemstelling

2.1.1 Situering van het probleem

De stage is begonnen met het eindproduct van een IT-project. Acht weken lang werd aan de beginselen van de Crap App gewerkt; hierbij werd bijvoorbeeld een pipeline aangemaakt om automatisch een build te kunnen genereren van de applicatie, of om andere zaken automatisch te laten uitvoeren. Zo werden ook geautomatiseerde testen opgezet om de Crap App te controleren op eventuele bugs of andere mankementen in de software. Deze geautomatiseerde testen vormen de centrale omgeving waarin deze stageopdracht zich afspeelt.

De geautomatiseerde testen van de Crap App werd in het IT-project opgesteld met als doel alle kritieke eigenschappen van de applicatie zeker te testen. Aangezien goede geautomatiseerde testen geen erg belangrijk aandachtspunt vormde bij het IT-project, werd de meeste tijd gestoken in het afwerken van andere opdrachten en componenten voor het IT-project. Op die manier werd er voornamelijk voor gezorgd dat de testen op zich werkten en werd er iets minder aandacht besteed aan het aanmaken van goed onderhouden geautomatiseerde testen. De gemaakte testen deden wat ze moesten doen, maar meer ook niet. De toekomst van de opgestelde testen en de opbouw ervan werden niet volledig in het achterhoofd gehouden.

Dit vormde de basis voor de probleemstelling van deze stageopdracht, namelijk het *future proof* maken van de geautomatiseerde testen en ervoor zorgen dat (nieuwe) testen opgesteld worden aan de hand van best practices.

2.1.2 Achtergrondinformatie

Om volledig mee te zijn in het verhaal van de Crap App, dient het uiteraard duidelijk te zijn wat de Crap App nu juist is. Kort samengevat is de Crap App een webbased applicatie die er op het eerste zicht uitziet als een normale applicatie. Ze bevat echter schakelbare bugs of softwaredefecten die kunnen opgespoord worden door beginnende testers. Op die manier kunnen junior testers binnen een veilige omgeving zichzelf trainen op het vinden van een brede waaier aan bugs, zonder dat de tester het risico loopt om bijvoorbeeld in een echt project iets te proberen dat catastrofale gevolgen kan veroorzaken.

Qua werking van het normale gedeelte van de applicatie, kan een gewone gebruiker of administrator een account aanmaken, zich inloggen op de applicatie, zijn account bekijken en eventueel details aanpassen, vrienden toevoegen en/of verwijderen en tot slot een forum raadplegen om met andere gebruikers in discussie te gaan.

Om toegang te krijgen tot het vitale gedeelte van de Crap App, kan een gebruiker surfen naar een speciale URL waarbij bugs aan- of uitgezet kunnen worden. Zodra de gebruiker teruggaat naar het ‘gewone’ gedeelte van de applicatie, kan hij beginnen met zijn zoektocht naar de verschillende bugs. Een bug kan een spelfout zijn waar gemakkelijk overeen gekeken kan worden, maar ook een beveiligingsfout die problemen kan veroorzaken in de database van de applicatie. De mogelijkheden zijn eindeloos.

2.1.3 Stakeholders

Elk project heeft personen of groepen wiens de mening een invloed kunnen hebben op (de uitkomst van) het eindresultaat. In de eerste plaats gaat dit om de mensen van Refleqt zelf. Aangezien Refleqt in hoofdzaak een bedrijf is dat zich focust op testing, zou een trainingsapplicatie een belangrijke bijdrage kunnen leveren aan de opleiding van beginnende testers.

Verder is de inspraak belangrijk van personen die de applicatie zouden kunnen gebruiken, namelijk mensen die nog geen tot weinig ervaring hebben met testing. Dit is wel een toeval, aangezien de auteur van dit eindwerk zelf nog niet veel ervaring had met testing. Persoonlijke meningen zijn dus van waarde binnen dit project.

2.2 Doelstellingen

De Crap App dient om beginnende testers in een veilige omgeving te laten kennismaken met het opsporen van bugs. Vandaar dat een goed opgestelde applicatie erg belangrijk is.

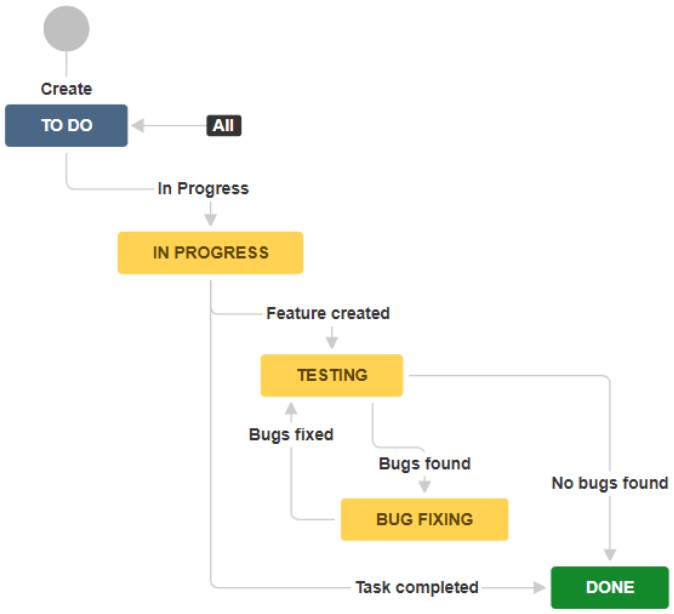
Deze stage brengt veel kennis bij over hoe dat testing verloopt in een echt project. Het opruimen van de eerder gecreëerde geautomatiseerde testen, het opstellen van mogelijk nieuwe testen, extra analyse die erbij komt kijken… stuk voor stuk zijn dit allemaal waardevolle taken om te volbrengen.

Qua onderzoek wordt een nieuwe manier van testtechniek onderzocht, namelijk *Model Based-Testing* (MBT). MBT is een begrip in de wereld van testing dat nog niet heel lang bestaat. Hierover een overzichtelijk onderzoek uitvoeren is absoluut van nut voor de stagiaire en zeker ook voor het stagebedrijf. Voor het bedrijf is dit een efficiëntere wijze om nieuwe informatie over het nieuwe begrip snel te begrijpen en verder door te geven aan interne werknemers.

2.3 Technologieën

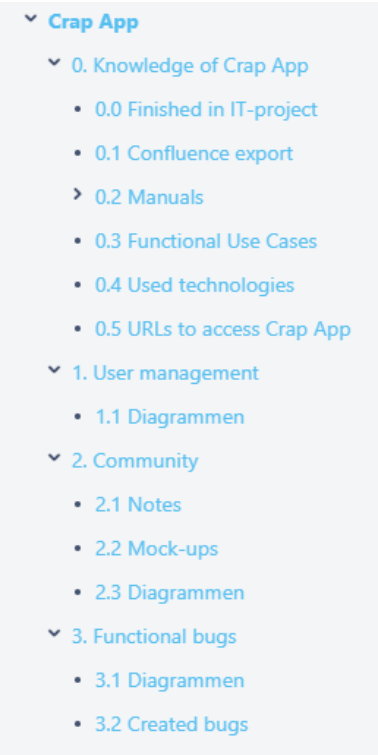
De stageopdracht wordt uitgevoerd met behulp van verschillende technologieën. Aangezien de analyse van toekomstige componenten voor de Crap App ook tot het takenpakket hoort, wordt in de eerste plaats van zowel Jira als Confluence gebruikgemaakt.

Jira is een ticketsysteem dat het leven van niet alleen IT-managers makkelijker maakt, maar eigenlijk van het hele development team. Het geeft de status van een project visueel en in realtime weer zodat het volledige team te allen tijde op de hoogte is van de stand van zaken. Tickets kunnen binnen Jira software gesleept worden naar een nieuwe status dankzij een voorop gestelde workflow. Zo kan een ticket opgenomen worden door een ontwikkelaar en *in progress* zijn, of kan een ticket volledig klaar zijn en dus op *done* staan. Dit betekent dat een ticket een heel proces heeft doorlopen vooraleer het mocht bestempeld worden als volledig afgewerkt. Hieronder is in figuur 2 te zien hoe de workflow in elkaar zit voor deze stage. [10]



Figuur 2: Workflow binnen Jira

Confluence komt Jira perfect tegemoet. Het is namelijk door hetzelfde bedrijf, Atlassian, ontwikkeld en zorgt voor een ideale aanvulling aan Jira software doordat Confluence als een krachtig verlengstuk voor Jira kan functioneren. De sterkte van Confluence is hoofdzakelijk het overzichtelijk bijhouden van veel (tekstuele) informatie. Zo is in figuur 3 hieronder te zien hoe de pagina's met informatie voor de Crap App zijn georganiseerd. Verder is het aanmaken van uitgebreide pagina's met behulp van krachtige templates slechts een van de vele mogelijkheden van Confluence. Bovendien kunnen Confluence en Jira zonder veel moeite met elkaar gekoppeld worden en bieden ze ontzettend veel mogelijkheden om verbindingen te leggen met duizenden andere tools. [9]



Figuur 3: Page tree binnen Confluence

Voor de het aanmaken van de geautomatiseerde testen wordt in de eerste plaats gebruik gemaakt van IntelliJ IDEA. IntelliJ IDEA is een ontwikkelomgeving van het bedrijf Jetbrains. Binnen het programma kan code geschreven kan worden en kunnen ook testen gemaakt worden. Verder biedt IntelliJ IDEA de mogelijkheid om te verbinden met een scala aan tools, zoals BitBucket, een tool die instaat voor Git-codemanagement. Hiermee kan code automatisch op een *version control system* gezet worden dat ervoor zorgt dat meerdere ontwikkelaars tegelijkertijd aan dezelfde code kunnen werken, zonder het risico op verlies van gegevens. Verder wordt code ook veilig opgeslagen en is de kans dat code verloren gaat beperkt tot een minimum. BitBucket is een van de vele tools dat gebruikmaakt van de technologie genaamd Git. Zoals al eerder uitgelegd, wordt het gebruikt om de code te waarborgen van dit project. [2], [3], [4]

Om ervoor te zorgen dat de Crap App op elk moment ergens kan gebruikt worden, zoals bij een seminarie of bij een (junior) tester op locatie, zorgt de technologie Docker voor de draagbaarheid van de applicatie. Dankzij Docker kan een instantie van de Crap App binnen een *container* overal op elk moment opgezet worden, mits een paar minuten instellen.

Tot slot zijn Selenium en Cucumber de twee tools die het testing gedeelte voor hun rekening nemen. Selenium staat in voor de automatisatie van browserhandelingen, zodat de computer bijvoorbeeld automatisch surft naar een website, iets aanklikt en vervolgens een tekstveld invult. Het gegeven voorbeeld is slechts het topje van de ijsberg qua mogelijkheden van Selenium. Daaropvolgend verzorgt Cucumber het schrijven van de testen zelf. Aangezien de effectieve code voor de testen geschreven is in Java, dient er ook een verbinding te zijn naar de testen zelf. De testen zijn in '*plain text'* geschreven: dit wil zeggen dat de testen op zich goed leesbaar zijn voor (zelfs) niet-technische mensen. Ze bestaan uit eenvoudige zinnetjes die een bepaalde actie in een test weergeven. Deze regels moeten ook verbonden worden met de nodige code zodat er ook effectief iets gebeurt bij het uitvoeren van de test: hiervoor komt Cucumber van pas. Vandaar dat Cucumber en Selenium in dit project onlosmakelijke met elkaar verbonden zijn; beide zijn nodig om succesvol testen uit te voeren.

3 Uitwerking stageopdracht

3.1 Beschrijving opdracht

3.1.1 Beginsituatie

Zoals al eerder aangegeven vormt de beginsituatie van deze stage het eindproduct van een afgerond IT-project. De gemaakte geautomatiseerde testen werken grotendeels en zijn er op voorzien om gewoon te werken. Het toepassen van best practices werd eerder op de achtergrond gelaten.

Een behoorlijk aantal testen was al opgesteld. Deze testen zijn erg nuttig voor het opstellen van de nieuwe testen, aangezien ze een werkend voorbeeld geven van hoe een test geschreven met Cucumber en Selenium exact werkt. Eerst wordt geleerd hoe zowel Cucumber als Selenium functioneren, en vervolgens wordt bekeken hoe de opbouw van de geautomatiseerde testen kan verbeterd worden. Dit vormt al snel het eerste grote obstakel en wordt in 3.1.4 besproken.

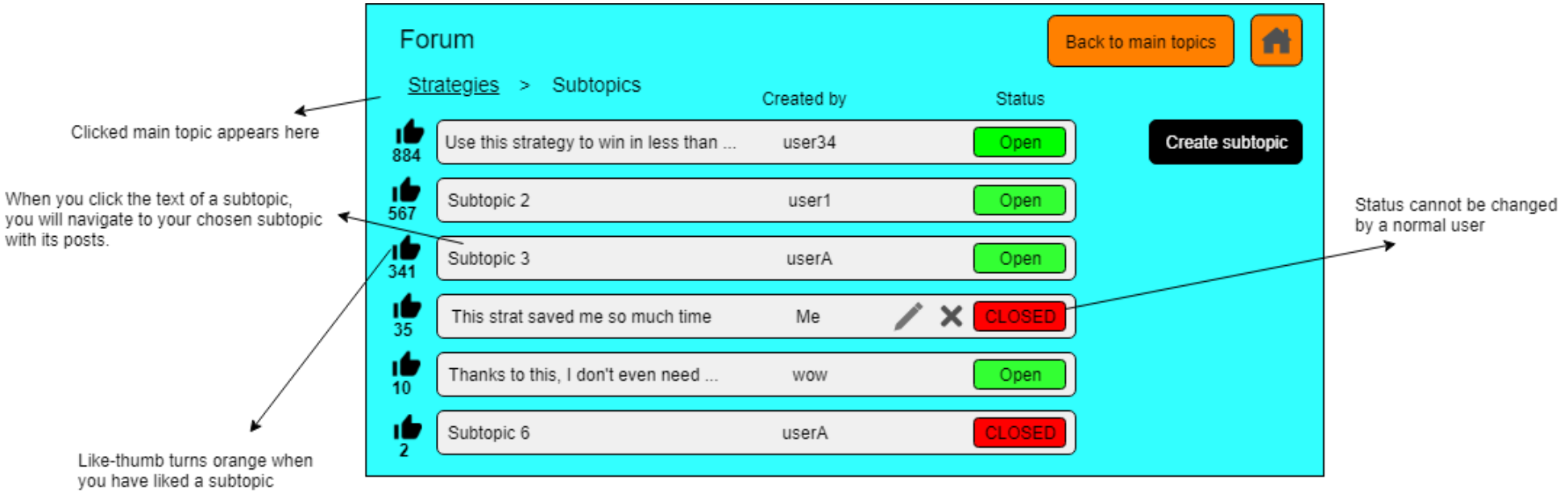
Langs de andere kant dient de Crap App verder uitgebreid te worden. Op het einde van het IT-project werd alvast een mooie basis gelegd, maar nieuwe componenten zijn zeker welkom in de applicatie. Om dit dus in goede banen te leiden, is naast het opstellen en opruimen van geautomatiseerde testen een bijkomende taak nodig, namelijk de analyse van nieuwe componenten en de management hiervan.

3.1.2 Analyse en management

Aangezien deze zijde van de stageopdracht geen onbekend terrein is, wordt dit gedeelte in beperkte mate besproken. Werken met Jira en Confluence of het opstellen van *activity diagrams* zijn nu eenmaal geen nieuwigheid.

Dit gedeelte bestaat uit drie algemene opdrachten: het inrichten en onderhouden van zowel Jira als Confluence, een analyse voor een nieuw component uitvoeren oftewel de diagrammen hiervoor opstellen en tot slot het aanmaken van mock-ups. Elk onderdeeltje wordt in dit hoofdstuk kort toegelicht.

Zoals bij hoofdstuk 2.3 Technologieën al wordt aangehaald, dienen zowel Jira als Confluence op punt gesteld te worden. Bij Confluence betekent dit dat er een logische documentenboom wordt samengesteld waaronder alle noodzakelijke documentatie snel en eenvoudig terug te vinden is. De informatie is per hoofdfunctionaliteit opgedeeld, met extra hoofdstukken voor oude informatie van het IT-project. Onder elke hoofdfunctionaliteit zijn telkens de diagrammen en mock-ups terug te vinden. Figuur 4 toont een mock-up van wat een gewone gebruiker ziet indien hij de subtopic pagina bezoekt van het Forum, een nieuw gedeelte van de Crap App.



Figuur 4: Mock-up van subtopic pagina

Jira is een ticketingsysteem dat de status van allerhande taken bijhoudt binnen een projectteam. Op een overzichtelijke manier kan elk teamlid kijken hoever een project staat, wie met wat bezig is en wat er nog gedaan moet worden. Elk ticket kan zich dus in een bepaalde status bevinden, de flow van deze statussen en de statussen zelf zijn ook aangemaakt voor dit project. Deze flow is te zien in hoofdstuk 2.3 als figuur 2.

Binnen Jira zijn de tickets opgesteld aan de hand van twee principes, enerzijds het MoSCoW-principe, anderzijds op basis van BDD, met behulp van de *Given When Then* methodologie. MoSCoW staat voor *Must have, Should have, Could have* en *Won't have*. Elk sleutelwoord geeft de prioriteit aan van een bepaalde feature. Een eis met *Must have* moet in het eindproduct zitten, anders wordt het product beschouwd als niet afgewerkt. Eisen met *Should have* zijn gegeerd, maar zonder deze eisen is het product wel nog bruikbaar. De derde prioriteit, *Could have*, zijn optionele eisen. Als er extra tijd gevonden wordt, kunnen deze features toegevoegd worden. Uiteindelijk blijft *Won't have* nog over, hierbij wordt aangenomen dat deze feature niet meer volbracht zal worden, maar kan het wel interessant zijn voor eventueel een toekomstig project. [32]

De *Given When Then* methode wordt in meer detail besproken in het volgende hoofdstuk, 'Algemene opbouw van een geautomatiseerde test'.

3.1.3 Algemene opbouw van een geautomatiseerde test

Om de opbouw van geautomatiseerde testen beter te begrijpen, wordt een algemeen idee geschetst van hoe individuele testen worden opgebouwd. Een eenvoudig voorbeeld wordt erbij genomen om de volledige flow van een individuele test uitgebreid te kunnen bespreken. De test gaat na of een gebruiker in staat is om vanaf de homepage te navigeren naar de profielpagina van de gebruiker zelf. Aangezien deze test van laag niveau is, is de test goed te volgen.

Na de analyse van een nieuwe component van de Crap App, wordt meteen bepaald hoe de test geschreven wordt. Er wordt namelijk beroep gedaan op het principe van *Behavior Driven Development* (BDD). BDD is een manier van ontwikkelen die de communicatie bevordert tussen zowel de technische als de businesszijde van een project. Een testscenario wordt opgesteld aan de hand van drie sleutelwoorden, genaamd *Steps*: *Given*, *When* en *Then*. Andere sleutelwoorden zoals Feature en Scenario dienen eerder om de context aan te geven van de test. Er zijn nog andere sleutelwoorden binnen Cucumber, maar om de uitleg helder te houden, blijft de uitleg voorlopig beperkt. Figuur 5 geeft het principe weer bij het gekozen voorbeeld, dit in een file genaamd UserProfiles.feature. De '@Sanity' annotatie mag voorlopig genegeerd worden. [5], [8]

```
@Sanity
Feature: User Profiles

  Scenario: Go to user profile from home page
    Given I am a logged in user-
    And I am on the home page
    When I click the profile div
    Then the browser navigates to my profile page
```

Figuur 5: Voorbeeld van Given, When en Then

Feature geeft op een erg korte wijze weer wat deze test inhoudt. Het geeft in theorie de documentatie weer van de specifieke test. Meestal blijft de uitleg bij dit sleutelwoord erg beknopt, maar het is mogelijk een extra beschrijving toe te voegen door verder te schrijven onder het sleutelwoord. Cucumber negeert deze tekst bij het uitvoeren van een tekst, maar om voldoende context te scheppen voor testers of andere leden van het development team, wordt deze beschrijving wel opgenomen in de rapporten die Cucumber genereert. Op die manier is het meteen duidelijk over welke test het gaat en wordt de kans op verwarring tot een minimum beperkt. Zodra er wordt gebruik gemaakt van andere Cucumber sleutelwoorden, eindigt de descriptie of documentatie van de test. [8]

Scenario vertelt wat de test eigenlijk doet. In een zin wordt de volledige uitvoer van de test beschreven. '*Go to user profile from home page*' geeft weer dat een ingelogde gebruiker zich bevindt op de h*ome page* en vervolgens navigeert naar de *profile page* van de gebruiker. *Scenario Outline* is een uitbreiding op het sleutelwoord Scenario. Het biedt de mogelijkheid om hetzelfde Scenario meerdere keren uit te voeren, maar met verschillende waardes. In de regels die beginnen met een Steps sleutelwoord, dat hieronder wordt uitgelegd, kunnen bepaalde woorden vervangen worden door een *placeholder* of tijdelijk woord met behulp van < en >, zoals <aantal>. Onder de testregels wordt vervolgens een extra stukje toegevoegd dat begint met '*Examples:*'. Figuur 6 geeft hiervan een voorbeeld weer. De eerste lijn woorden komt overeen met de placeholder woorden in de test zelf. Elke lijn onder de eerste lijn woorden geeft een nieuwe variatie van het scenario weer. Deze test wordt dus in essentie twee keer uitgevoerd met telkens verschillende waarden. [8]

```
Scenario Outline: eating
  Given there are <start> cucumbers
  When I eat <eat> cucumbers
  Then I should have <left> cucumbers

  Examples:
    | start | eat | left |
    |    12 |   5 |    7 |
    |    20 |   5 |   15 |
```

Figuur 6: Voorbeeld van een Scenario Outline

De *Steps* sleutelwoorden zijn de regels binnen een test die een aparte stap van de test beschrijven. Deze sleutelwoorden bestaan uit *Given*, *When*, *Then*, *And* of *But*. Deze vormen samen het hart van de test. [8]

Given geeft de voorwaarden aan waaraan de beginsituatie van de test moet voldoen om de test te kunnen laten slagen. In dit geval dient een gewone gebruiker ingelogd te zijn op de Crap App én aanwezig te zijn op de homepage van de gebruiker. Indien er meerdere voorwaarden van toepassing zijn, kan er om de leesbaarheid te verbeteren gebruikgemaakt worden van een extra sleutelwoord, genaamd *And*. *And* geeft de tester de ruimte om een extra zinnetje toe te voegen op elke plaats, dus niet alleen bij *Given*. Zoals al eerder aangehaald, verhoogt dit de leesbaarheid van de test op zich en zorgt dit voor een makkelijkere implementatie van code bij elke testregel. Er moet per slot van rekening op een gegeven moment code vastgemaakt worden aan elke testregel. In dit geval wordt er dus later code vastgemaakt aan elke regel die begint met een van de sleutelwoorden. [5], [8], [13]

When geeft een actie, event of trigger weer. Zodra de test zich in de correcte omstandigheden bevindt om de test uit te voeren, kan een actie of event meegegeven worden die in principe de trigger vormt van de test. In het voorbeeld is dit het klikken op de *profile div*, een onderdeel van de webpagina dat de gebruiker naar zijn *profile page* brengt indien hij erop klikt. Aangezien deze actie de trigger vormt van deze test, wordt ervoor gezorgd dat de test verder wordt uitgevoerd. [5], [8], [13]

Then geeft de finale situatie weer, hierop gebeurt de uiteindelijke controle of de test effectief geslaagd is of niet. Het geeft het verwachte resultaat weer. Indien dit verwachte resultaat effectief gebeurt bij het uitvoeren van de test, is de conclusie dat de test geslaagd is. Bij het voorbeeld wordt verwacht dat de browser uiteindelijk navigeert naar de *profile page* van de gebruiker. Waarom wordt hier aangegeven dat specifiek de browser navigeert naar de *profile page*? Omdat de gebruiker zijn stappen om te kunnen navigeren perfect kan uitvoeren, maar als de navigatie naar de *profile page* niet correct gecodeerd is, is uiteindelijk de browser niet in staat om te navigeren, in tegenstelling tot de gebruiker. Aangezien deze regel op deze manier beschreven is, is het ook meteen duidelijk waar de fout zou kunnen liggen indien de test faalt bij deze *Step*. [5], [8], [13]

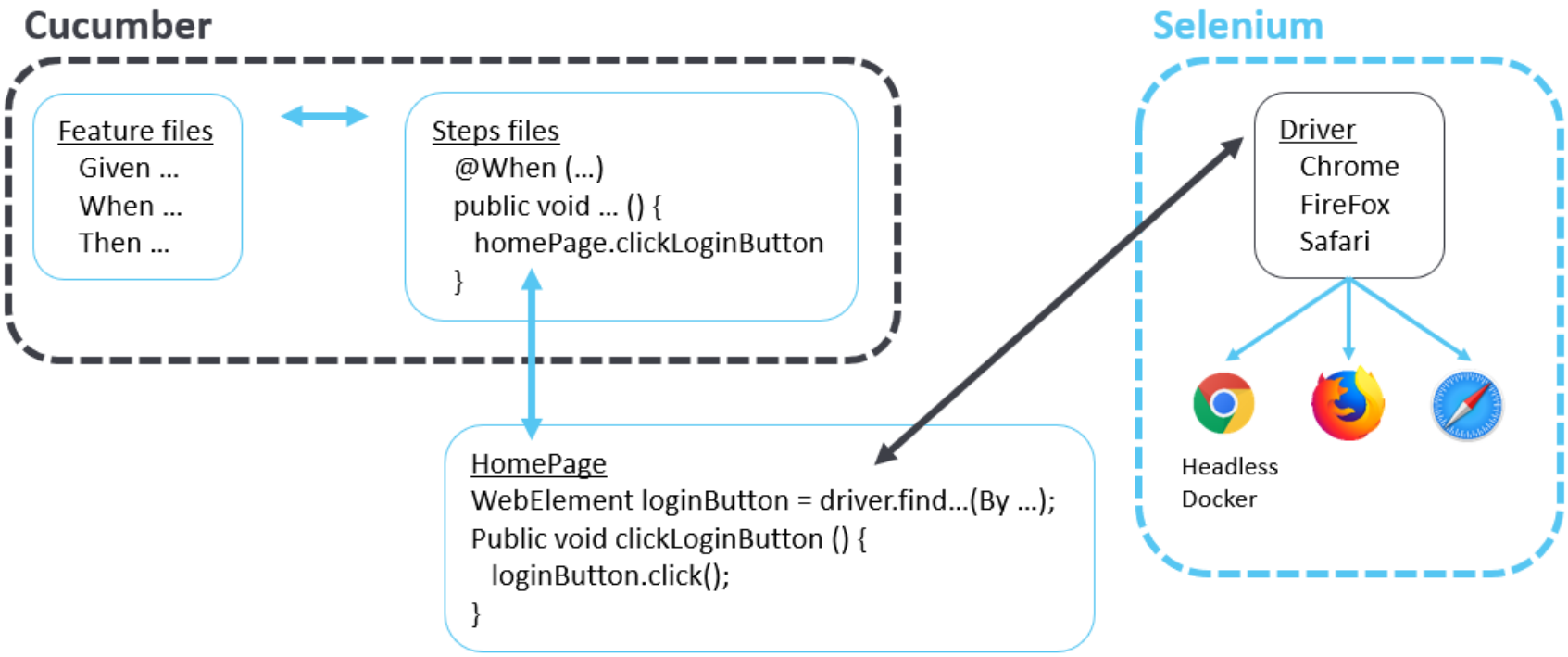
3.1.4 Opruim geautomatiseerde testen

3.1.4.1 Algemene werking geautomatiseerde testen

Geautomatiseerde testen hebben verschillende onderdelen nodig die elk een tandwiel vormen in een groot proces, samen maken alle onderdelen het mogelijk om uiteindelijk een of meerdere testen te laten uitvoeren. Zo goed als alle delen van deze geautomatiseerde testen zijn nieuwe kennis, dus geeft figuur 7 een (zelfgemaakt en) kort overzicht weer van de noodzakelijke elementen. Niet alle onderdelen zijn vernoemd, in de figuur staan de meest belanghebbende elementen waaraan iets gewijzigd wordt, of waarvan de betekenis erg belangrijk is. Gedetailleerde uitleg over specifieke klassen of andere behulpzame functionaliteiten worden in hoofdstuk 3.1.4.2 verder toegelicht.

Het valt meteen op dat de figuur is opgedeeld in drie domeinen. Een Cucumber gedeelte, een Selenium gedeelte en een gedeelte dat daar tussen valt. Om de werking van deze domeinen te begrijpen, wordt de opbouw uitgelegd van links naar rechts.

Ten eerste komt het gedeelte van Cucumber aan bod. Eerst worden zogenaamde feature files opgesteld met het de *Given, When* en *Then* sleutelwoorden. In hoofdstuk 3.1.3 wordt uit de doeken gedaan op welke manier deze sleutelwoorden zorgen voor het begin van een test. Aan elke unieke regel binnen een feature file wordt code gehangen, zodat wat beschreven wordt in die regel, ook effectief uitgevoerd kan worden als een werkende test. Deze code bevindt zich in *Steps* files. Zoals de naam van deze soort files al verraadt, wordt binnen deze files uitvoerbare code verbonden met de stapjes, *Steps* of unieke regels binnen feature files. Stel dat zo een regel bepaalt dat een gebruiker klikt op de login button op de home page. De code om dit te laten gebeuren wordt gehangen aan de testregel door een annotatie vlak voor de methode te zetten die deze functionaliteit in code beschrijft, in de figuur is dit @When.



Figuur 7: Opbouw geautomatiseerde testen Crap App

Natuurlijk is het noodzakelijk om te kunnen interageren met knoppen, velden en meer op een webpagina. Vandaar dat er dus wordt gebruik gemaakt van een POM. POM staat voor *Page Object Model*. Dit gedeelte bevindt zich tussen Cucumber en Selenium en vormt als het ware een brug tussen beide aspecten. Het idee van POM is om van elke unieke webpagina waarop bijvoorbeeld testen uitgevoerd kunnen worden, een bruikbaar object te maken. Elk interactief element op een webpagina wordt als een *webelement* gedefinieerd in code, zoals de login button. Een methode wordt aangemaakt binnen de code van de klasse van de home page dat ervoor zorgt dat er geklikt kan worden op deze knop. Om uiteindelijk effectieve browseracties zoals kliks te kunnen uitvoeren, dient elk *webelement* wel verbonden te zijn met een *driver*.

Drivers zijn een onderdeel van Selenium. Zoals al eerder wordt toegelicht, maakt Selenium het mogelijk om effectief een klik uit te voeren zonder tussenkomst van een mens, of om bijvoorbeeld een veld in te vullen. Selenium maakt de cyclus dus compleet om een geautomatiseerde test te kunnen uitvoeren.

Wat is nu juist een *driver*? Een *driver* is een instantie van een webbrowser, zo kan een *chromeDriver* gebruikt worden, maar ook van bijvoorbeeld FireFox of Safari kunnen *webdrivers* aangemaakt worden. Doordat er bij elke nieuwe uitvoering van een test een nieuwe driver wordt aangemaakt, is het mogelijk om deze driver bepaalde argumenten mee te geven, zoals gemaximaliseerd opstarten of *headless* opstarten. Als het *headless* argument wordt meegegeven, ziet de tester niets op zijn scherm gebeuren. Het uitvoeren van de test(en) gebeurt op de achtergrond en neemt dus geen extra GPU-geheugen in. Dikwijls is het wel erg handig om dit argument achterwege te laten, zodat een tester kan zien hoe de test verloopt. Indien er iets onverwacht fout loopt, kan een tester op visuele wijze ontdekken wat of waar iets de mist ingaat.

Verder is het woord Docker terug te vinden in de figuur. Zoals in het hoofdstuk Technologieën wordt toegelicht, staat Docker in voor de draagbaarheid van de Crap App. Binnen een container van Docker kan een instantie van de Crap App opgestart worden om zo met de applicatie te werken of om er testen op uit te voeren. Om te weten of opgestelde testen daadwerkelijk werken, is het dus cruciaal om de applicatie werkend te kunnen laten draaien.

3.1.4.2 Specifieke werking geautomatiseerde testen

Aangezien bepaalde specifieke elementen van de geautomatiseerde testen volledig nieuwe informatie zijn, worden deze in dit hoofdstuk kort besproken.

In *Steps* files worden per unieke webpagina de testregels voorzien van code. Soms komt het voor dat een of meerdere regels regelmatig worden herbruikt, zoals bijvoorbeeld het aanwezig zijn op de home pagina. In plaats van dit telkens opnieuw te definiëren in een nieuwe *Steps* file, kunnen deze vaak voorkomende regels samengenomen worden in een gemeenschappelijke *Steps* file, in het geval van dit project heet deze file commonSteps. Dit zorgt uiteindelijk voor het minder schrijven van code en het efficiënt toepassen van gemeenschappelijke code.

Hetzelfde principe kan toegepast worden voor page objecten, maar in het geval van dit project is er niet genoeg code die in elke webpagina opnieuw gebruikt wordt. Daarom is er dus geen commonPage file opgesteld.

Voor de algemene ondersteuning van alle nodige files zijn twee hulpklassen aangemaakt, SupportFunctionalities en DataProvider. De eerste is nog een overblijfsel van het oorspronkelijke IT-project met enkele methoden die mogelijk nuttig zijn voor de toekomst. Hierin wordt de hoofdinstantie van de driver aangemaakt. Via DataProvider is de driver te bereiken voor andere files die gebruik willen maken van de driver. Het feit dat alle files aan slechts één instantie kunnen van de driver, is erg belangrijk. Erg veel problemen hadden als oorzaak problemen met de driver, of eigenlijk het feit dat er meerdere driver tegelijkertijd werden toegepast, wat zorgde voor conflicten. Voor de rest zorgt DataProvider voor een vaste gewone gebruikersaccount en een admin account en verleent de klasse toegang tot alle URL's van de Crap App.

Naast algemene ondersteuning, is ook nood aan extra functionaliteit voor de testen op zich. Zo is CustomListener opgesteld. Deze klasse zorgt ervoor dat bepaalde events worden afgevuurd vlak voor of vlak na een bepaalde actie. In het geval van de Crap App wordt dit toegepast op de volgende manier: na elke actie, dus bijvoorbeeld na een klik, wordt een screenshot genomen en wordt deze opgenomen in het rapport dat Cucumber uiteindelijk opmaakt.

Een andere functionaliteit bevindt zich in de klasse Hooks. De werking van Hooks is te vergelijken met CustomListener. Hooks bevat een beforeHook en een afterHook. Code die meegegeven wordt binnen deze hooks, worden respectievelijk altijd voor of na een testscenario uitgevoerd. Dit verschilt met CustomListener doordat CustomListener code uitvoert voor of na elke specifieke actie. Een testscenario is een volledige doorloop van één test met één bepaalde set van gegevens. Binnen een scenario kunnen meerdere acties voorkomen, zoals kliks, dus is het mogelijk dat de code van CustomListener meerdere keren binnen een scenario wordt uitgevoerd. Bij elk nieuw scenario wordt de code uit Hooks telkens opnieuw ervoor of erna uitgevoerd.

Bij de Crap App wordt Hooks gebruikt om het huidige scenario op te slaan en om de huidige driver af te sluiten. Aangezien beide bij elk scenario 'ververst' dienen te worden, is het dus noodzakelijk gebruik te maken van Hooks.

3.2 Resultaat

Een grote test die alle functionaliteit nagaat van de loginpagina, is het resultaat van deze stageopdracht. Elk onderdeel van de loginpagina wordt in dit hoofdstuk nagegaan, samen met hoe dit precies wordt getest. Het aanklikken van knoppen vergt geen speciale handelingen op vlak van testen, maar bij bijvoorbeeld het invullen van tekstvelden ligt dit anders. Hieronder volgt een overzicht van screenshots die elk iets meer vertellen over de test.

Figuur 8: Loginpagina van de Crap App

Figuur 8 toont de loginpagina. Net als de vier functionaliteiten die hier zijn aangeduid, bestaat de test ook uit vier grote onderdelen. Eerst wordt de registratie besproken; het klikken op *register* brengt de gebruiker naar een nieuwe webpagina, deze is te zien in figuur 9. Bij de registratie van een nieuwe gebruiker is een waaier aan informatie nodig. Op vlak van testen zijn er vier tekstvelden te controleren, een groepje radiobuttons en twee knoppen. Om elke mogelijke situatie te checken, zijn meerdere scenario's nodig.

Figuur 9: Registratiepagina van de Crap App

Omdat het aantal screenshots anders dramatisch zou stijgen, is alle code van de grote test terug te vinden in bijlage B. Om toch nog op een overzichtelijke manier de verschillende stukken binnen LoginPageFunctionality.feature te bespreken, wordt gewerkt met een extra aanduiding in de bijlage. Het aangeduide deel B2 geeft bijvoorbeeld het gedeelte over de registratie weer. De code wordt best bekeken in bijvoorbeeld IntelliJ IDEA of Notepad++.

De onderdelen van de test worden chronologisch besproken, te starten met B1. B1 vormt het begin van elk scenario. Elke keer als een nieuw scenario wordt uitgevoerd, met al dan niet een nieuwe set van gegevens, start de test vanaf het begin. Dit begin wordt vastgelegd in de *Given* regel. In het geval van deze test, wordt altijd begonnen op de loginpagina. *Background* is nog een sleutelwoord van Cucumber. De enige functie van *Background* is om context aan testers te verschaffen. De tekst bij *Background* is geen code die uitgevoerd kan worden, het geeft eenvoudigweg mee waarover de test gaat.

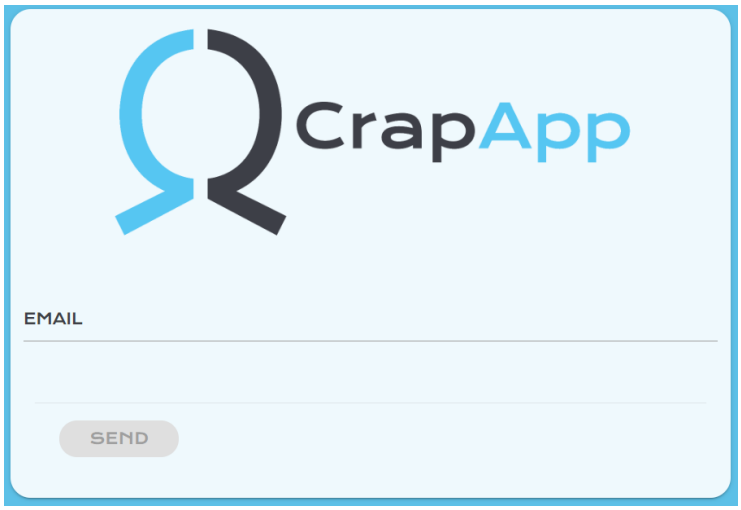
Het volgende gedeelte, B2, controleert de registratiefunctionaliteit. Doordat meerdere velden gecontroleerd moeten worden, zijn meerdere scenario's nodig om elke situatie na te gaan. Dit kan gedaan worden op de traditionele manier, namelijk door elk scenario volledig uit te schrijven, maar aangezien dit veel extra werk oplevert, wordt hier gebruik gemaakt van een *Scenario Outline*. Bij een *Scenario Outline* wordt eerst het algemene verloop van de verschillende scenario's beschreven, namelijk de verschillende velden worden telkens ingevuld en er wordt op de knoppen geklikt. Wat er precies wordt ingevuld of op welke knoppen telkens geklikt wordt, dat wordt in het *Examples* gedeelte bepaald.

In de *Scenario Outline* wordt gebruik gemaakt van vervangbare worden, zoals <email>. Deze termen kunnen vervangen worden door een gegeven uit de kolom met dezelfde naam uit *Examples*. Voor het eerste scenario wordt bijvoorbeeld een 'juist' pad doorlopen. Een geldig e-mailadres wordt ingevuld samen met een geldige gebruikersnaam, mannelijk wordt als geslacht aangeduid en een juist wachtwoord met een overeenkomend wachtwoord wordt tot slot ingevuld. Elke lijn van *Examples* vormt op die manier een scenario. Bij de registratie zijn er dus 9 scenario's opgesteld.

Als deze scenario's zijn afgerond, mag nog een functionaliteit binnen de registratie niet vergeten worden, namelijk de cancel button. Een klein scenario hiervoor controleert deze knop.

B3 bevat het gedeelte over de login functionaliteit. Ook bij het inloggen van een gebruiker kunnen allerlei situaties voorkomen. Om te proberen de meeste mogelijkheden te controleren, wordt opnieuw gebruik gemaakt van een *Scenario Outline*. Dit keer zijn er heel wat minder velden of knoppen te controleren, maar kan wel heel wat uiteenlopende informatie meegegeven worden. Zo kan de gebruikersnaam juist ingevuld worden, fout ingevuld worden of worden leeggelaten. Hetzelfde geldt voor het wachtwoord, maar kan het wachtwoord volgens de richtlijnen worden ingevuld en niet bij de gebruiker passen, of wordt iets ingevuld dat niet voldoet aan de vereisten van een correct wachtwoord. Hoe dat deze velden ingevuld worden, geeft verschillende foutberichten weer. Als deze fouten correct getoond worden, slagen de verschillende scenario's.

Als een gebruiker zijn wachtwoord vergeet, moet er ook iets voorzien worden zodat deze gereset kan worden. In B4 is te zien hoe dit getest wordt. De gebruiker navigeert naar een pagina zoals in figuur 10 te zien is. Aangezien hier maar een veld en een knop aanwezig zijn, is uitgebreide testfunctionaliteit niet nodig. Een kleine *Scenario Outline* test het email veld en een fout wordt al dan niet getoond.



Figuur 10: Webpagina van vergeten wachtwoord/gebruikersnaam

Bij B5 is een erg gelijkaardig proces te zien, met als verschil dat de gebruiker zijn gebruikersnaam vergeten is. Aangezien dit proces bijna identiek is aan het vorige proces, wordt dit niet meer in detail besproken.

4 Reflectie

Het begin van deze stageopdracht viel me erg zwaar. De Crap App was een project met een stevige basis, alleen erg veel verfijning en een goede opruim van het project ontbrak. Bestaande componenten hadden nood aan optimalisering en de testen dienden opgefrist te worden. Een nieuw gedeelte voor de applicatie was ook gewenst. Kortom, er kwam erg veel op mij af.

Mijn mede-stagiair, Floriaan Zandijk, kreeg de taak mij bij de brengen met de stand van zaken, aangezien hij meegewerkt had aan het IT-project rond de Crap App. Floriaan deed zijn uiterste best, maar het begrijpen van de verschillende beslissingen van het project, of de principes van testing die werden toegepast, bleek een veel grotere uitdaging dan verwacht. Het kostte mij dus ontzettend veel tijd om het verhaal volledig te begrijpen, met name hoe testing binnen dit project in zijn werk ging. Zo ben ik langzaam maar zeker kunnen beginnen aan het verbeteren van de oorspronkelijke geautomatiseerde testen. Deze testen waren eerst opgesteld om gewoon te werken, het toekomstperspectief werd eerder naar de achtergrond geschoven. Dankzij mijn eigen ervaringen met mijn IT-project, had ik hier volledig begrip voor. Maar om iets te kunnen verbeteren, moest ik het eerst volledig verstaan. Het verstaan van het testen kostte mij dus bijzonder veel tijd. De opruim en optimalisatie werd dus een proces dat ook erg langzaam vooruitgang boekte.

Tegelijkertijd was het ook de bedoeling een nieuw gedeelte toe te voegen aan de Crap App. Eerst zouden er meerdere componenten toegevoegd worden, maar door mijn trage vooruitgang bleef het uiteindelijk bij één nieuw component, een forum. Algemene management van het project heb ik op mij genomen, samen met de analyse van het forum. Ik heb Jira klaargestoomd voor dit project, Confluence aan onze noden aangepast en voor de analyse heb ik ook gebruik gemaakt van deze tools. Het opstellen van tickets, modellen en mock-ups nam ik op terwijl ik ook werkte aan de opruim van de geautomatiseerde testen.

Uiteindelijk ben ik hierdoor geraakt en kon ik beginnen aan het aanmaken van de grote testen. Deze testen zouden elk de functionaliteit van elke unieke webpagina nagaan, maar tijdens het werken aan een test besefte ik dat ik het bij één grote test zou moeten houden. Na overleg met Wim werd besloten dat dit goed was zodat ik mijn eindwerk op een mooie manier kon afsluiten.

Als ik hierop terugkijk, krijg ik het gevoel niet veel gerealiseerd te hebben. Ik ben een trage leerling, dus de tijd zat niet met me mee. Langs de andere kant heb ik wel erg veel bijgeleerd over testing binnen BDD. Ik heb leren werken met bijvoorbeeld Docker, een technologie waarvan ik nooit dacht dat ik er effectief iets mee zou doen. Uiteindelijk ben ik wel trots op waar ik nu sta. Dit had ik nooit gedacht in het begin van mijn stageperiode.

II. Onderzoeksonderwerp

1 Onderzoeksvraag

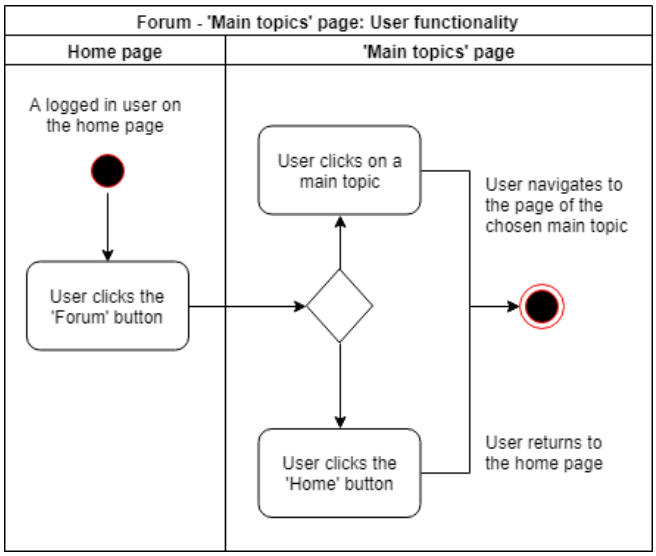
Binnen het technologische milieu is testing geen nieuwigheid meer. Na het manueel testen van software en later applicaties, blijven de mogelijkheden van testing evolueren. Hoewel testen voor een hogere kwaliteit zorgt, blijft het enorm veel tijd kosten. Zo wordt er continu gezocht naar efficiëntere manieren om testen (meermaals) uit te voeren, of zelfs om software volledig automatisch te laten testen na een eenvoudige druk op de knop. Geautomatiseerd testen wordt al snel een opkomende trend in het testing milieu en zelfs dit blijft nog steeds verder evolueren. Een volgend hoofdstuk in het verhaal van testen, zou wel eens de naam '*Model Based-Testing'* (MBT) kunnen krijgen.

De bedoeling van het onderzoek van deze stagebundel is om de betekenis en werking te achterhalen van het concept van MBT. Hoe kan MBT van waarde zijn voor Refleqt? Dit wordt verder onderzocht door te kijken welke tools er op de markt zijn en welke tool(s) eventueel zouden kunnen worden toegepast op de stageopdracht zelf, de opstelling van de geautomatiseerde testen voor de Crap App. Aangezien MBT een omvangrijk onderwerp is, wordt de focus gelegd op tools die werken met BPMN, zodat het aantal tools dat onderzocht wordt, beperkt wordt tot een minimum. De betekenis van BPMN wordt verder toegelicht in het volgende hoofdstuk: methode van onderzoek.

2 Methode van onderzoek

Aangezien *Model Based-Testing* een volledig nieuw concept is, zal de eerste fase van het onderzoek bestaan uit exploratief onderzoek. Met dit soort onderzoek wordt bedoeld dat er eerst wordt geprobeerd een weg te zoeken in het verhaal van *Model Based-Testing*. Centraal hierbij wordt getracht het gekozen onderwerp beter te begrijpen en mogelijk interessante bronnen te verzamelen om deze later in het onderzoek te kunnen raadplegen. Kortom, in deze fase van het onderzoek wordt een algemeen beeld gevormd van het begrip *Model Based-Testing*. [11]

Na een eerste inschatting kan er doelgerichter gewerkt worden en wordt deskresearch aangenomen als het volgende type onderzoek. De onderzoeksvraag, samen met de deelvragen, maakt dit iets eenvoudiger; zo vraagt Refleqt een vergelijking te maken tussen verschillende *Model Based-Testing* tools die beschikbaar zijn op de markt, met de focus op BPMN, een standaard voor procesmodellering. BPMN staat voor *Business Process Modeling Notation*. Het is een manier om processen op een visuele manier weer te geven aan de hand van diagrammen. In figuur 11 is een diagram te zien van een van de functionaliteiten van de Crap App. Dit diagram is een *activity diagra*m, het geeft de mogelijke acties weer van een stakeholder bij een bepaald gedeelte van de applicatie. In dit geval is te zien wat een gewone gebruiker kan doen indien hij zich op de hoofdpagina bevindt van het forum van de Crap App. [11]



Figuur 11: Voorbeeld van een activity diagram, gemaakt met BPMN-technieken

Bij deskresearch komen twee activiteiten naar voren, betrouwbare informatie over de bestaande tools verzamelen en informatie inwinnen van Refleqt. Betrouwbare informatie zal hoofdzakelijk online teruggevonden worden, met als reden dat informatie op een online platform dikwijls het meest actueel en dus correct is. Software wijzigt voortdurend, dus is het belangrijk gebruik te maken van de meest recente en uiteraard betrouwbare bronnen. Verder mag de informatie vanuit Refleqt zeker niet vergeten worden. Refleqt geeft aan welke eigenschappen van een *Model Based-Testing* tool belangrijk zouden zijn voor Refleqt zelf. Op basis hiervan kan er uiteindelijk bepaald worden welke tool het beste voor Refleqt zou kunnen werken.

Deze eigenschappen om beide tools mee te vergelijken kunnen worden gebruikt in een vergelijkingsmatrix. Deze methode van vergelijken werd geleerd bij het vak *Customer/Business relationship management* (CRM). De precieze werking van deze matrix wordt toegelicht in hoofdstuk 3.2 binnen deel twee van dit eindwerk.

Om de cirkel rond te maken, wordt het onderzoek afgesloten met een toegepast perspectief. Betrouwbare bronnen worden geraadpleegd, informatie wordt ingewonnen bij Refleqt zelf en het theoretisch onderzoek wordt afgerond. Als slotstuk wordt een praktisch gedeelte toegevoegd om het verhaal af te maken. Dit zal bestaan uit toegepast onderzoek. Hierbij wordt een experiment opgezet en wordt de functionele zijde van de winnaar(s) van het vorige gedeelte geverifieerd. Een *proof of concept* wordt opgezet voor ofwel de tool die overduidelijk naar voren kwam als beste, ofwel voor meerdere tools. Afhankelijk van de uitkomst van de theoretische vergelijking van de tools, wordt de redenering verder uitgebreid waarom een bepaalde tool een betere keuze is, of wordt een uiteindelijke winnaar gekozen tussen de verschillende *Model Based-Testing tools*. [11]

3 Resultaten

3.1 Literatuurstudie

3.1.1 Geschiedenis

Bij het ontwikkelen van software is het testen van de gemaakte software een noodzakelijk goed. Ook al wordt een programma ontwikkeld door ’s werelds beste programmeurs, bugs of fouten in software zullen altijd de kop opsteken. Vandaar dat testen absoluut niet achterwege gelaten mag worden bij de volledige *Software Development Life Cycle* (SDLC). Dankzij het regelmatig testen van software, kan niet alleen de tijd die nodig is om een product af te leveren sterk worden verminderd, ook de kwaliteit van het product zelf zal alleen maar stijgen. Echter, het vraagt tijd om degelijke geautomatiseerde testen met goede testmethoden op te zetten.

Manueel testen is de eerste vorm van controle voor een stukje software. De ontwikkelaar voert zijn code uit en gaat na of de code doet wat hij moet doen. Op het eerste zicht lijkt alles misschien in orde, maar bugs kunnen zich verbergen op de meest uiteenlopende plaatsen. Zo kan een oneindige lus het volledige werkgeheugen van een computer opslorpen en wordt dit pas duidelijk als het eigenlijk te laat is. Het grote probleem van manueel testen is het verlies van tijd. Aangezien alle functionaliteit van een programma telkens opnieuw gecontroleerd moet worden, om zeker te zijn dat alles nog werkt zoals het hoort, kruipt hier erg veel tijd in. Als een tester elk testscenario bij elke update van een programma moet doorlopen, worden ontzettend veel middelen gestoken in het eenvoudigweg controleren of een programma nog werkt. Kortom, het manueel testen van software bleek al snel niet meer genoeg controle te zijn om voldoende kwaliteit te garanderen van een product. Te veel tijd gaat verloren aan adequaat testen en dit kost erg veel geld. [7]

Een natuurlijke opvolging van het voorgaande probleem wordt het automatiseren van testen. Vooral testen die dikwijls herhaald moeten worden om te kunnen garanderen dat bijvoorbeeld een bepaalde functie blijft werken, kunnen in theorie gemakkelijk geautomatiseerd worden. Dankzij onder andere scripting en nieuwe tools die op de markt worden gebracht, wordt het meer en meer toegankelijk om van testautomatisatie een vaste waarde te maken. Natuurlijk vraagt het tijd om te leren werken met deze nieuwe technologieën, maar zeker op lange termijn zal niet alleen de kennis van de tester op vlak van geautomatiseerd testen toenemen, ook de kwaliteit zal stijgen, aangezien meer getest kan worden op minder tijd. Een zekere investering hiervoor is dus van groot belang bij de eerste fase van een project. Voldoende middelen, zoals trainingen of onlinecursussen zijn hier goede voorbeelden van. [5]

Geautomatiseerd testen heeft ook een evolutie doorgemaakt en evolueert nog steeds. Het prille begin van een niet-manuele test was het maken van een script voor een manueel script. Dit klinkt misschien nogal tegenstrijdig, maar het simuleren van eenvoudige teststappen vormde al snel een eerste script waarbij de computer de stappen van een testproces voor de tester zelf deed. Het aanklikken van een knop, het navigeren naar een webpagina, een tekstveld invullen… Stuk voor stuk werden eenvoudige handelingen op een computer omgezet naar code. Hieruit groeide al snel de behoefte naar tools die het de tester makkelijker maken om handelingen op een computer te automatiseren. Extra functionaliteit werd al snel toegevoegd, zoals het genereren van rapporten met testresultaten of de mogelijkheid om in te plannen wanneer testen uitgevoerd worden. Zo kunnen testen ’s nachts draaien en kan overdag de tester aan de slag met de resultaten, ook dit bevordert de efficiëntie van de beschikbare werktijd van een tester. [5]

Net als een methodologie sterk in populariteit groeit, neemt de behoefte naar tools met meer specifieke eigenschappen toe. Zo ontstaan verschillende soorten frameworks die elk een andere manier toepassen van geautomatiseerd testen, zoals *Linear Scripting Frameworks*, *Modular Testing Frameworks*, *Data Driven Testing Frameworks*, *Keyword Driven Testing Frameworks*, *Hybrid Testing Frameworks* en tot slot *Behavior Driven Development* (BDD) *Testing Frameworks*. Binnen de stageopdracht wordt gewerkt met een BDD framework. [12]

Maar wat is nu juist een framework? Een framework is, binnen de context van geautomatiseerd testen, een verzameling van regels en best practices om een geautomatiseerd project op poten te kunnen zetten. Elk uniek framework volgt een vaste groep van regels en best practices dat aansluit bij een bepaalde manier van geautomatiseerd testen. Het hangt dus van het framework af hoe de werking van het framework er nu precies uitziet. Om een beeld te geven op het soort framework dat wordt toegepast binnen de stageopdracht, wordt hieronder een korte beschrijving gegeven van BDD Frameworks. [12], [14]

De centrale gedachte van een BDD framework is om alle betrokkenen van een project te kunnen laten participeren in zowel de development als het testen van het uiteindelijk gemaakte project. Een brug maken tussen zowel business als IT blijft tot op de dag van vandaag nog steeds een uitdaging, vandaar dat dit framework in populariteit wint. In plaats dat er op puur technische wijze gekeken wordt naar de noden van een programma, worden test specificaties opgesteld in 'natuurlijke taal'. Dit wilt zeggen dat de specificaties ook te begrijpen zijn door personen van de businesszijde van een project. De technologie die binnen de stageopdracht wordt toegepast, is Cucumber. [12], [14]

Om terug te komen op het verhaal van MBT, is MBT geen framework op zich. Het is een nieuwe techniek van geautomatiseerd testen. Wat MBT nu precies inhoudt, wordt in het volgende hoofdstuk verklaard.

3.1.2 Definitie

MBT is de huidige evolutie op vlak van testing. In plaats van elk stapje in een test apart te automatiseren en eerst nog testcases apart op te stellen, zorgt MBT ervoor dat deze stappen bijna volledig overgeslagen kunnen worden. Om een goede definitie te bepalen, wordt in eerste instantie een research paper van Vasudha Singh en Subburaj Ramasamy onder de loep genomen. [1]

Subburaj Ramasamy is een gerenommeerde professor op vlak van IT. Hij heeft jaren ervaring met programmeren op verschillende vlakken en zijn doel is om kwaliteit overal te verbeteren. Samen met Vasudha Singh, een doctoraat student Toegepaste Psychologie, heeft hij een korte maar krachtige research paper geschreven over MBT. Vandaar dat zijn paper over MBT zeker een waardevol element toevoegt binnen dit onderzoek over wat MBT nu eigenlijk is. [1], [7]

De paper beschrijft een korte exploratie van MBT, met name wat er precies bij komt kijken en hoe een project met MBT aangepakt zou kunnen worden, een discussie over verschillende types van modellen en hoe het testen aangepakt kan worden met deze modellen. Verder komt een case study aan bod dat gebruik maakt van MBT en tot slot een conclusie. Deze bron wordt mogelijk meerdere malen geraadpleegd, aangezien Subburaj Ramasamy een gerenommeerde professor is op vlak van IT, onder andere met testing. Vandaar dat de paper als bijlage A te raadplegen is. [1]

Een quote rechtstreeks uit het artikel van beide onderzoekers, geeft een eerste beeld op wat MBT inhoudt.

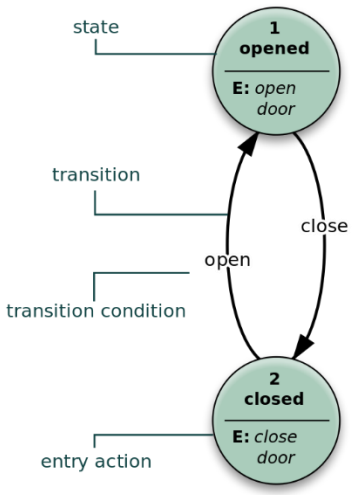
"*MBT [2] is the generation of software test procedures, using models of system requirements and behavior. In MBT we create model of system under test (SUT) and generate test cases from the model*." [1]

Na het ontleden van deze definitie, zijn er drie sleutelwoorden die naar de voorgrond springen, met name *generation*, *models* en testcases. De meest belangrijke functionaliteit van MBT bestaat dus uit het genereren van testcases op basis van modellen. Elk sleutelwoord vormt een cruciaal onderdeel van MBT en is nodig om het principe van MBT te laten slagen. Om te trachten dit beter te begrijpen, wordt elk sleutelwoord apart geanalyseerd en uiteindelijk met elkaar in verband gebracht. *Models* geeft het startschot.

3.1.2.1 Modellen

Binnen MBT, wat wordt er precies bedoeld met *models* of modellen? Volgens het artikel is een softwaremodel een weergave van het gedrag van de software. Het gedrag van software bestaat uit een breed scala van sequenties, met name input voor het systeem, zoals acties en voorwaarden. Output logica of de flow van data doorheen de componenten van de software horen ook thuis onder mogelijk gedrag van software. Om deze grote verzameling aan opties visueel weer te geven in de vorm van een model of diagram, dient deze informatie uit context getrokken te worden. Dit wilt zeggen dat het model van het gedrag van software moet worden opgesteld op de meest verstaanbare manier, zonder inmenging van kennis van mensen die al gewerkt hebben de software in kwestie. In theorie zou elke persoon van het team in staat moeten zijn het model te begrijpen, ook al heeft de persoon nog geen ervaring met de desbetreffende software. Als het model is opgesteld op een heldere manier, wordt het niet alleen herbruikbaar en deelbaar, maar ook de meest precieze omschrijving van de *system under test* (SUT). Een SUT is het systeem dat getest wordt. Dit kan gaan over de gehele applicatie, maar ook over een specifiek gedeelte van een programma. Aangezien wat getest wordt binnen een project op bijna continue basis wijzigt, wijzigt bijgevolg de inhoud van de SUT ook voortdurend. [1]

Modellen bestaan in verschillende vormen. Om niet te ver af te dwalen van het onderwerp, worden twee soorten modellen besproken. Zo is een van de mogelijke types modellen een *finite state machine* (FSM) model. Dit soort diagram geeft weer in welke staat de SUT zich bevindt en hoe de SUT in die staat terecht kan komen. Een eenvoudig voorbeeld kan dit verduidelijken. Een cd-speler kan uitgeschakeld zijn, ingeschakeld zijn of bezig zijn een cd af te spelen. Een mogelijke flow in dit geval zou het volgende kunnen zijn; de cd-speler staat uit, een gebruiker drukt op de *power button* van het apparaat en de cd-speler is ingeschakeld. Vervolgens steekt de gebruiker een cd in het apparaat en drukt hij op *play*. De cd speelt af en muziek is te horen. Een FSM-model geeft de paden weer om van en naar elke staat te kunnen gaan. Om dit nog beter te illustreren, wordt natuurlijk best een effectief diagram getoond. Figuur 12 geeft een erg eenvoudig FSM-model weer, namelijk over de staten waarin een deur zich zou kunnen bevinden en hoe de deur naar de andere staat kan overgaan. [1], [15]



Figuur 12: Voorbeeld van een eenvoudig FSM-model [15]

Unified modeling language (UML) komt ook aan bod in het artikel, dit is eerder een verzameling van modellen. UML is dus niet een soort diagram, het bestaat uit verschillende soorten diagrammen die complex gedrag of eigenschappen van software visueel kan weergeven. Specifiek werd UML ontworpen met als doel objectgeoriënteerde analyses en ontwerpen voor informatiesystemen te maken, dit op grafische wijze. Binnen UML zijn diverse modellen gegroepeerd, waaronder het eerder besproken *state* diagram. Verder bestaat er ook een indirecte connectie met BPMN, een standaard voor procesmodellering op business niveau. Een *activity diagram*, een type diagram dat ook onder UML valt, lijkt erg sterk op procesmodelleringstechnieken van BPMN. Soms kunnen beide technieken tegelijkertijd toegepast worden omdat ze zo sterk op elkaar lijken, dit is het geval bij bepaalde diagrammen van de stageopdracht. Er werden *activity diagrams* opgesteld voor nieuwe componenten van de Crap App, deze werden gemaakt aan de hand van BPMN. In het onderdeel van de stageopdracht wordt deze diagrammen meer toegelicht. [1], [16]

3.1.2.2 Generatie van testcases

Aangezien de sleutelwoorden generatie en testcases onlosmakelijk met elkaar verbonden zijn, worden ze in dit hoofdstuk samen besproken. De algemene generatie van testcases gebeurt aan de hand van het model dat gebruikt wordt. Elk mogelijk pad binnen het model wordt beschouwd als een testcase. Omdat een computer in plaats van een mens het model inspecteert en alle mogelijke paden eruit haalt, is de kans veel kleiner dat er opties gemist worden. Menselijke fouten vormen nog steeds een risico, dus dit risico tot een minimum beperken kan alleen maar in het voordeel van het projectteam werken. Bovendien spaart dit tijd en dus geld uit, dus mogen de voordelen van MBT absoluut niet onderschat worden. [1]

De manier waarop de testcases worden gegenereerd, hangt eigenlijk af van welke MBT tool gebruikt wordt. Er zijn namelijk verschillende strategieën die het genereren van testcases op een andere manier aanpakken. Deze benaderingen worden in hoofdstuk 3.1.3 toegelicht. [1]

3.1.3 Strategieën

Zoals de definitie van MBT in hoofdstuk 3.1.2 aangeeft, draait MBT om het automatisch genereren van testcases op basis van een model. Dit klink zowel erg voor de hand liggend, alsook erg vaag. Wat wordt precies met generatie bedoeld? Op het eerste zicht lijkt het logisch dat hiermee elke mogelijke doorloop van een model bedoeld wordt. Voor kleine en zelfs middelmatig grote modellen zou deze aanpak kunnen werken, maar bij erg grote en complexe modellen zou dit al snel een probleem vormen. Een tester wilt waarschijnlijk niet bij elke sessie testing alles opnieuw testen. Soms wilt een tester een bepaald gedeelte van een model testen of werken met bepaalde parameters om zo te bepalen welke testen van cruciaal belang zijn. Om deze behoefte van testers tegenmoet te komen, maken sommige MBT-tools gebruik van benaderingen of strategieën voor het genereren van testcases die aansluiten bij de noden van de tester.

Om een idee van de verschillende strategieën te krijgen, worden twee bronnen geconsulteerd. De eerste bron is een pdf van het bedrijf Conformiq. Conformiq biedt een MBT-tool aan genaamd Conformiq Creator, die in hoofdstuk 3.2.4 toegelicht wordt in een vergelijking van drie MBT-tools. De pdf schetst een beeld van waarmee rekening gehouden moet worden bij het selecteren van een MBT-tool. Een onderdeel van deze bron beschrijft een aantal aanpakken van MBT. De tweede bron wordt in het volgende hoofdstuk ook gebruikt en is terug te vinden in bijlage D. Het omschrijft namelijk specifiek over de vele vormen van strategieën bij MBT. [37], [38]

Conformiq houdt het erg beperkt met drie mogelijke strategieën of manieren van aanpak voor MBT: *graphical test modeling*, *environment model-driven test generation* en *system model-driven testen generation*. Volgens Conformiq zijn dit de drie voornaamste manieren van aanpak voor MBT. Het algemene doel van deze benaderingen blijft hetzelfde, namelijk het genereren van testcases met eventueel bijhorende documentatie. Het verschil zit juist in efficiëntie; hoeveel werk moet een tester verrichten vooraleer testcases gegenereerd kunnen worden? [38]

De eerste benadering bedraagt het grafisch verwerken van testcases. Het is de meest eenvoudige strategie. Hierbij worden de testcases gepresenteerd op visuele wijze. In plaats dat een tester naar bijvoorbeeld een hoop tekst moet kijken om te zien wat voor testcases nu juist gegenereerd zijn, ziet de tester meteen welke testcases aangemaakt zijn. Een beeld zegt tenslotte veel meer dan duizend woorden. [38]

Ten tweede beschrijft Conformiq *environment model-driven test generation*. Hierbij vertellen de modellen hoe de *system under test* (SUT) wordt gebruikt en hoe de omgeving rondom het systeem werkt. Dit klinkt nogal verwarrend, maar hiermee wordt bijvoorbeeld de tester bedoeld. Deze strategie geeft eigenlijk bijkomende informatie voor de tester terug, zoals suggesties hoe het systeem getest kan worden, welke inputs best gebruikt worden en bijpassende checks op vlak van output. Als de tester deze informatie zelf moet bekomen, is hij bijzonder veel tijd kwijt. Via deze manier van aanpak voor MBT krijgt de tester op maat gemaakte informatie over het effectief testen van de software. Het is mogelijk dat dit toch niet 100% aansluit bij de volledige werking van de SUT, maar het geeft de tester tenminste alvast een idee van hoe het systeem getest kan worden. Dit spaart opnieuw tijd en geld uit. [38]

De laatste strategie voor MBT is volgens Conformiq *system model-driven generation*. Het verschil met de vorige aanpak, is dat hier het model het effectieve, wenselijke gedrag van het systeem voorstelt. [38]

Aangezien Conformiq op vlak van strategieën niet meer veel informatie levert, wordt nu overgegaan naar de tweede bron, namelijk het onderzoeksdossier dat effectief onderzocht welke strategieën er zijn voor MBT: '*A taxonomy of model-based testing approaches*'. [37], [38]

De paper in bijlage D bekijkt zes dimensies van verschillende MBT strategieën en vergelijkt ze op vlak van verschillende tools die van deze strategieën gebruik maken. Voor het onderzoek van dit eindwerk wordt enkel naar de theoretische zijde van de strategieën gekeken. De opdeling in dimensies komt voort uit een opdeling die hieraan vooraf gaat, namelijk *model specification*, *test generation* en *test execution*. Binnen *model specification* bevinden zich *scope*, *characteristics* en notaties. Binnen *test generation* zijn *test selection criteria* en *technology* terug te vinden en *test execution* bevat *on/offline*. Wat achter elke dimensie schuilt, wordt hieronder toegelicht. [37]

De eerste dimensie gaat over model scope. Slechts een vraag is hier belangrijk, zijn alleen inputs belangrijk voor de SUT, of het volledige input en output gedrag van de SUT? Hoewel werken met enkel input veel makkelijker is, zorgt het ook voor meer beperkte informatie. Zo kunnen gegenereerde tests niet als een zogenaamd orakel gebruikt worden; tests waarmee voorspeld kan worden hoe een systeem zich gedraagt onder bepaalde omstandigheden. Werken met alleen input creëert dus geen toekomstbestendige tests. Indien zowel input als output kan worden opgenomen in het model, is het mogelijk dat de tests wel als een sterk orakel kunnen optreden. In het kort zorgt een combinatie van zowel input als output voor informatie over de omgeving waarin de tests gebeuren, alsook informatie over de SUT zelf. [37]

Characteristics van een model vormen de tweede dimensie. Afhankelijk van hoe de SUT eruit ziet, wordt bepaald welke eigenschappen te pas komen bij het te gebruiken model. Zo zijn drie eigenschappen bepalend voor het model. Allereerst speelt tijd een rol. Zo zijn er verschillende systemen die in realtime draaien; ook deze systemen moeten (continu) getest worden. Dit vormt nog steeds een uitdaging voor MBT. Verder wordt het concept van indeterminisme aangehaald. Indeterminisme betekent dat gebeurtenissen zonder reden gebeuren en dat er een zekere vrije wil is. Volgens bijlage D kan het principe van indeterminisme ingezet worden bij modellen om bepaalde gedragingen te controleren. Bepaalde functies van een applicatie worden vaker gebruikt dan anderen en dus dienen die functies al dan niet meer getest te worden. Zo is het mogelijk dat testen opgesteld worden op basis van eerdere reacties van de SUT. Er kan (onbewust) prioriteit gegeven worden aan testen waarvan de SUT ze meer van belang vindt dan andere situaties. Dit kan nuttig zijn, maar het kan ook zorgen voor niet genoeg test coverage. Vandaar dat dit principe twee uitkomsten heeft waarmee rekeningen gehouden moet worden. Tot slot is er ook nog sprake van de dynamiek van het model. Modellen kunnen discreet of continu zijn, of zelf een combinatie van beide (hybride). Systemen die discreet te werk gaan, worden bijvoorbeeld door events geactiveerd. Systemen die continu werken, blijven functionaliteiten uitvoeren in realtime. Zoals al eerder wordt vermeld, vormt het testen van dit soort systemen nog steeds een uitdaging. [37], [39]

Dimensie drie van de MBT strategieën gaat over verschillende notaties van het model. Bijlage D beschrijft ten eerste *state-based notations*, oftewel een model van de staat van een systeem op een bepaald moment. Transition-based notations leggen de aandacht op de transities van een systeem. Een voorbeeld hiervan zijn *finite state machines*. De volgende notatie bedraagt *history-based notations*. Hierbij wordt het gedrag van het systeem over een periode van tijd gemodelleerd. Verder bestaan er *functional notations*. Deze vorm van notatie maakt gebruik van wiskundige functies. *Operational notations* omschrijft systemen als een verzameling van uitvoerbare processen die gelijktijdig lopen, zoals bij communicatie services. Verder beschrijven *stochastic notations* een systeem op basis van de waarschijnlijkheid. Hierbij wordt vooral berust op input data en worden bijgevolg voornamelijk modellen opgemaakt voor de omgeving van de testen. Als laatste blijven de data-flow notations over. Bij deze notatie ligt de focus op data. Samengevat, elke notatie biedt een antwoord op de verschillende noden van een tester of van het te testen systeem. Voldoende onderzoek dient hiervoor worden gedaan. [37]

De vierde dimensie volgens bijlage D is *test selection criteria*. Er zijn verschillende criteria die de selectie van de testen kunnen beïnvloeden. Zo kan de structuur een grote rol spelen, deze gaat meestal hand in hand met de derde dimensie. Indien er gekozen wordt voor een transitie gebaseerd model, wordt gebruik gemaakt van alle transities, alle cyclussen, alle verbindingen, enzovoort. Een tweede criteria focust op *data coverage*. Hierbij wordt gekeken met welke data nu getest moet worden, aangezien een systeem wel eens gebruik maakt van een enorme database. Een methode die wordt toegepast, is het uitkiezen van data dat een beeld geeft van een bepaalde categorie. Op die manier wordt (hopelijk) alle data gerepresenteerd. Een andere criteria kijkt eerder naar de *requirements coverage*. Alle of specifieke requirements dienen getest te worden. *Ad-hoc test case specifications* zijn een volgende criteria. Hierbij beslist de tester expliciet welke testcases effectief getest dienen te worden. Niet alle paden van een model zijn altijd even nuttig om te testen. Verder is er nog *random and stochastic criteria*. Dit wordt bij omgevingsmodellen toegepast, er worden namelijk onverwachte manieren van testen gegenereerd. Tot slot blijft *fault-based criteria* over. *Fault-based criteria* zoekt specifiek naar fouten in de SUT. [37]

De technologieën van test generatie vormt de vijfde dimensie van MBT-benaderingen. Technologieën zoals het willekeurig genereren van testcases, het gebruik maken van *search-based* algoritmes of het controleren van een model zijn allemaal mogelijkheden. *Symbolic execution* voert een model uit met een bepaalde set van input data, oftewel zogenaamde *constraints* of beperkingen. Het model moet rekening houden met de opgelegde limieten. [37]

De laatste dimensie kijkt naar test executie. Zoals onder andere bij de definitie van MBT en het kritiek perspectief wordt bepaald, maakt test executie niet expliciet deel uit van MBT. Sommige tools zijn in staat de gegenereerde test cases uit te voeren, maar zeker niet alle tools bieden dit aan. Om terug te komen op de zesde dimensie, wordt de uitvoering van de testcases online ofwel offline gedaan. Door online te testen, kunnen outputs van de SUT als uitkomst uit de bus komen. Ze reflecteren een realistisch beeld. Offline zijn de testcases strikt opgesteld. Realistisch gedrag komt mogelijk niet voor. Langs de andere kant kan dankzij offline testing er meteen opgevolgd worden met test management tools. Een aanpassing aan het test proces is snel gebeurd. Een set van testen kan een keer gegenereerd worden en deze kunnen meerdere keren opnieuw uitgevoerd worden. Verder kunnen deze testen op verschillende machines uitgevoerd worden, alsook in andere of nieuwe omgevingen of op andere tijdstippen. Een belangrijke opmerking bij dit deze dimensie is dat de testen zowel manueel als automatisch uitgevoerd kunnen worden. Beiden hebben voor- en nadelen. Een menselijk oog dat een systeem bekijkt kan mogelijk veel meer ontdekken dan een computer die op voorhand bepaalde testen uitvoert. Een computer zal eigenaardig gedrag dat niets met zijn huidige test te maken heeft, niet rapporteren. Een mens is wel veel trager en maakt dikwijls fouten. Beide manieren hebben dus zijn nut. [37]

Samengevat schildert dit hoofdstuk een beeld van de erg complexe zijde van de verschillende manieren waarop MBT toegepast kan worden. Er zijn ontzettend veel methoden om MBT af te stemmen op hetgeen dat getest moet worden. Dit grote aanbod zorgt dus voor verwarring bij een tester. Gelukkig besloten sommige MBT-tools te kiezen voor een eigen aanpak. De bron van Conformiq lijkt een erg basic beeld te schetsen van de mogelijke strategieën van MBT, maar dat is met rede gedaan. Conformiq biedt namelijk een eigen MBT-tool aan en maakt dus in principe reclame voor de werking van de strategieën. Dezelfde strategieën komen terug in de tool. Door de extreme keuzevrijheid bij de klant of tester weg te nemen, maakt Conformiq het des te makkelijker de klant te laten kiezen voor hun product. De bron uit bijlage D maakt dan weer duidelijk dat er erg veel opties zijn, te veel opties mogelijk. Hierin duidelijkheid scheppen is een uitdaging en blijft materie voor de toekomst. [37], [38]

3.1.4 Kritiek perspectief

Aangezien de meeste informatie in hoofdstuk 3 wordt gehaald uit het artikel in bijlage A, is het van cruciaal belang dat de gevonden informatie onder de loep wordt genomen. Hier en daar zijn extra bronnen geraadpleegd voor het verifiëren van stukjes informatie, maar dat is niet genoeg om de tekst voldoende beargumenteerd te bestempelen. Vandaar wordt binnen dit hoofdstuk extra aandacht besteed aan het nagaan of de aangenomen definitie weliswaar klopt.

Voor deze kritieke blik worden twee bronnen geraadpleegd. Ten eerste wordt informatie gehaald uit een researchwerk dat onderzocht hoe de technieken van MBT het ervan afbrengen indien het wordt toegepast op projecten met miljoenen lijnen code. Ten tweede wordt een werk geraadpleegd dat zich meer focuste op de verschillende aanpakken binnen MBT. Zoals in hoofdstuk 3.1.3 wordt toegelicht, zijn er namelijk verschillende strategieën voor het genereren van test cases. De werken zijn beide als bron aangeduid in de bibliografie en zijn terug te vinden als respectievelijk bijlage C en D. [36], [37]

Beide bronnen schenken aandacht aan verschillende aspecten van MBT. Binnen dit hoofdstuk wordt hoofzakelijk gekeken of de definitie van MBT overeenkomt met deze bronnen.

Om even terug te komen op de oorspronkelijke definitie van bijlage A, luidt deze als volgt: MBT is het genereren van software test procedures aan de hand van modellen, systeem requirements of gedragingen van het systeem. De centrale gedachte van deze definitie is dus dat zogenaamde test procedures of testcases worden gegenereerd door modellen van de te testen software. [1]

Bijlage C beschrijft het principe van MBT als volgt.

> *"This approach involves developing and using a data model to generate tests. The model is essentially a specification of the inputs to the software, and can be developed early in the cycle from requirements information."* [36]

Volgens bijlage C maakt MBT gebruik van een data model voor het genereren van tests. De algemene gedachte over MBT is dus gelijk voor zowel de originele bron als bijlage C. De quote gaat wel iets dieper in op het model dat van toepassing is bij MBT. Het model geeft alle inputs van de software specifiek weer. Het kan al vroeg ontwikkeld worden in het proces van softwareontwikkeling, zo vroeg als bij het vergaren van informatie van requirements. Deze gedachtegang komt niet volledig overeen met de redenering van bijlage A. Hoofdstuk 3.1.2.1 geeft aan dat het model best het gedrag van (een gedeelte van) software voorstelt. Hiermee bedoelt bijlage A dat uit een model af te leiden is wat er gebeurt indien binnen een programma voor een bepaald pad is gekozen. Kiest een gebruiker ervoor de radio aan te zetten, is de gebruiker nu in staat het volume in te stellen, de radio weer uit te zetten, enzovoort. Volgens de eerste definitie van bijlage C kunnen binnen een model enkel de inputs van software terug te vinden zijn. [1], [36]

Deze uitspraak zorgt mogelijk voor verwarring; wat wordt namelijk bedoeld met de 'inputs van software'. Blijft deze manier van werken dan niet erg beperkt? Sectie 2 van bijlage C gaat hier gelukkig grondiger op in. Sectie 2 beschrijft onder andere notaties van het data model en test selectie algoritmes. De model notatie of de manieren hoe een model voor MBT ingevuld wordt, daarop wordt ingezoomd. [36]

Allereerst bespreekt bijlage C een opdeling van op welk niveau een model toegepast wordt. MBT kan namelijk ingezet worden op allerlei levels van testing. Het laagste niveau controleert de werking van een eenvoudige software module. Door de input parameters van deze module in kaart te brengen, kan al snel een groot aantal testen opgesteld worden. Dankzij de eenvoud van de (kleine) software module is

de generatie snel gebeurd en bevordert het de efficiëntie van de ontwikkelaars. Dit komt overeen met het centrale idee van MBT. [36]

MBT bij een applicatie van medium grootte gaat over het controleren van simpele gedragingen. Bijlage C noemt dit een *single step* in een applicatie. Dit gaat dus een stapje verder dan het vorige niveau. Met een *single step* wordt een enkelvoudige functie bedoeld, zoals het invoegen van een rij in een tabel, een bericht versturen of een formulier invullen. Qua complexiteit valt de generatie van testen voor dit soort applicatie goed mee. Net als het vorige niveau is de data set helemaal niet ingewikkeld en is dus het data model geen moeilijke materie. [36]

Uiteindelijk zijn er ook grotere applicaties die bij meer realistische scenario's aansluiten. Dit zijn dikwijls programma's die al op punt staan waarbij mogelijk hier en daar aanpassingen worden doorgevoerd. Hoewel dit een grotere uitdaging voor MBT vormt, brengt MBT op dit niveau het meeste op. Hierbij wordt opnieuw gedrag van een systeem gecontroleerd, maar op veel ingewikkelder niveau. Bijlage C noemt dit *flow testing*. De volledige flow van een programma doortesten leunt ook sterk aan bij het testen zoals een gebruiker of klant het zou doen. Een klant checkt niet of één knop werkt, een klant gaat door de volledige applicatie om zijn vonnis uit te kunnen spreken. Aangezien *flow testing* bijzonder veel tijd kost, zou het een enorm voordeel kunnen zijn indien MBT dit zou kunnen overnemen. Bijlage C beschrijft dit proces als een combinatie van het vorige niveau, namelijk *step testing*. Door meerdere steps samen te nemen, kan het testgedrag van een gebruiker gesimuleerd worden. Dit resulteert opnieuw in een model dat eigenlijk het gedrag van software controleert, zoals in bijlage A wordt aangegeven. Door beide bronnen ook qua oorsprong te vergelijken, is alvast een interessante tussentijdse conclusie te trekken. [36]

Hoewel beide bronnen uiteindelijk tot dezelfde constatering komen over het model van MBT, is het toch eigenaardig dat de bron van bijlage C dit pas later concludeert. Het feit dat bijlage C de twee niveaus vóór het laatste en meest interessante niveau zelfs bespreekt, is ook opmerkelijk. Dit heeft te maken met de oorsprong van beide bronnen. De eerste bron (van bijlage A) is gepubliceerd in 2015. Deze bron is dus nog vrij recent en omschrijft MBT is meer recente omstandigheden. De bron van bijlage C is gepubliceerd in 1999. Dat is een ruim verschil van 16 jaar in verschijning. Aangezien technologie erg snel evolueert, is het dus geen verrassing dat de visie op MBT ook sterk veranderd is. Vandaar dat bijlage C het belangrijk vond om de verschillende niveaus van testing te bespreken, terwijl tegenwoordig de eerste twee niveaus niet meer van belang zijn. Verrassend of niet, beide bronnen komen tot dezelfde conclusie. [1], [36]

Tot slot wordt een derde bron benaderd, deze is terug te vinden in bijlage D. Deze bron is afkomstig uit 2011 en sluit mogelijk aan bij de definitie van de bron uit bijlage A. Of dit zo is, wordt hier nagegaan.

Bijlage D geeft een eerste definitie van MBT, het luidt als volgt.

> *"Model-based testing (MBT) is a variant of testing that relies on explicit behaviour models that encode the intended behaviours of a SUT and/or the behaviour of its environment. Test cases are generated from one of these models or their combination, and then executed on the SUT."* [37]

Op het eerste zicht lijkt deze definitie inderdaad aan te sluiten bij de oorspronkelijk bekomen definitie. Er ligt een sterke nadruk op modellen die het gedrag van een systeem beschrijven. Verder wordt een stukje toegevoegd dat verklaart dat de gegenereerde testcases ook uitgevoerd worden. Dit is nieuw voor een definitie over MBT. Tot nu bestond de centrale gedachte meestal uit het automatisch genereren van testcases op basis van een model, maar ook het effectief uitvoeren van deze testcases is nog niet voorgekomen. [37]

Na het onderzoeken van verschillende MBT-tools, is het mogelijk om te bepalen of de uitvoer van testcases al dan niet bij de definitie van MBT hoort. Dit geeft als resultaat dat het niet hoort tot bij de kern van de definitie van MBT. Het genereren van de testcases behoort wel tot de kern van de definitie. Het effectief uitvoeren is eerder optioneel te verkrijgen bij een aantal tools, niet bij alle tools. In de industrie wordt het principe van MBT dus begrepen als puur de generatie van testcases, de uitvoer wordt als een optie gezien. [21], [22], [23], [37]

Kortom, de kern van MBT berust op de automatische generatie van testcases met behulp van een model. Dit model beschrijft het gedrag van het te testen systeem. [1], [36], [37]

3.2 Vergelijkingsmatrix

Het eerste gedeelte van het onderzoek bedraagt een voornamelijk theoretische aanpak. De achtergrond van het begrip MBT wordt toegelicht om zo een idee te geven wat het precies inhoudt. Om hierop verder te bouwen, vormt het volgende onderdeel van het onderzoek een vergelijking van drie tools die elk MBT toepassen, weliswaar op telkens een andere manier. Zoals al eerder werd aangegeven, zijn er ontzettend veel manieren waarop testcases gegenereerd kunnen worden, vandaar dat er dus ook verschillende tools op de markt zijn die elk een eigen specialiteit hebben.

Op algemeen vlak bestaat er dus waarschijnlijk geen 'beste' tool, maar voor Refleqt bestaat er misschien wel een tool die goed of zelfs perfect aansluit bij de noden van het bedrijf. Elk bedrijf heeft verschillende wensen en om er achter te komen welke tool het meeste zou kunnen bieden voor Refleqt, wordt gebruik gemaakt van een vergelijkingsmatrix.

3.2.1 Werking

Een vergelijkingsmatrix is een manier om tools te vergelijken op een zo objectief mogelijke wijze. Er worden drie (of vier) versies opgesteld van de vergelijkingsmatrix. Elke iteratie van de tabel geeft telkens een duidelijker beeld over welke tool het beste zou kunnen passen bij het bedrijf. De eerste versie van de vergelijkingsmatrix bedraagt een opsomming van gewenste eigenschappen of *requirements* en een eenvoudige visuele weergave of de *requirement* aanwezig is binnen de mogelijke tools. Bijgevolg krijgt deze versie van de tabel de naam Requirements Matrix. Deze *requirements* zijn (mogelijke) eigenschappen van een tool die voor het bedrijf in kwestie van belang zijn. Zo kan de kostprijs een mogelijke factor zijn; voor sommige bedrijven dient de prijs zo laag mogelijk te liggen, terwijl de prijs voor andere bedrijven niet meteen een grote rol speelt.

In tabel 1 is een sterk vereenvoudigde versie van de Requirements Matrix te zien. Per opgegeven *requirement* is via een kleurencode te zien of het aanwezig is binnen een bepaalde tool.

Tabel 1: Eenvoudige weergave Requirements Matrix

	Tool 1	Tool 2	Tool 3
Requirement 1	Beschrijving 1: requirement zit in de tool	Beschrijving 2: requirement is niet aanwezig in de tool	Beschrijving 3: requirement is deels beschikbaar, mogelijk bijkomende kosten of software vereist

Groen geeft weer dat de *requirement* volledig aanwezig is binnen de tool, oranje duidt op een gedeeltelijke aanwezigheid met mogelijk bijkomende kosten of noodzakelijke software en rood geeft aan dat de *requirement* volledig afwezig is binnen de tool. Binnen de gekleurde vakjes wordt bovendien een korte beschrijving gegeven van hoe de *requirement* verwerkt is binnen de tool.

Hoewel deze tabel visueel erg duidelijk laat zien wat nu wel en niet aanwezig is binnen een tool, geeft het toch nog geen volledig helder beeld. In dit geval is slechts een eigenschap te zien en blijft het overzicht behouden, maar in realistische gevallen gaat het al snel over tien of zelfs meer verschillende *requirements*. De wensen van een bedrijf kunnen erg specifiek zijn en hier dient rekening mee gehouden te worden. Op die manier zou een Requirements Matrix al snel niet meer overzichtelijk zijn en meer twijfel brengen dan klaarheid. Vandaar dat de mogelijkheden uitgebreid worden in de volgende iteratie van de vergelijkingsmatrix.

In de tweede versie van de vergelijkingsmatrix, genaamd de Score Matrix, blijven de kleuren behouden met dezelfde betekenis, maar wordt een score toegevoegd. Per *requirement* kan nu een numerieke score toegekend worden. Het getal in de kolom 'Score' heeft twee betekenissen. De eerste betekenis dient vanuit een breder perspectief bekeken te worden, in vergelijking met andere *requirements*. Het geeft aan hoe belangrijk de *requirement* is voor het bedrijf. Sommige *requirements* zijn dikwijls van groter belang dan anderen, dus door meer belangrijke *requirements* een hogere maximale score te geven, zoals in het geval van tabel 2, wegen deze *requirements* uiteindelijk zwaarder door in de finale conclusie van de vergelijksmatrix. Minder beduidende eigenschappen wegen minder zwaar door en hebben dus minder kans om een doorslaggevend resultaat terug te geven. Bij tabel 2 is de maximale score 100 meer belanghebbend dan de maximale score 50. De scores van de eerste *requirement* zijn dus bepalender dan de scores van de tweede *requirement*. Over de berekeningen van de definitieve resultaten wordt later meer verteld.

Tabel 2: Eenvoudige weergave Score Matrix

	Score	Tool 1	Tool 2	Tool 3
Requirement 1	100	80	30	100
Requirement 2	50	50	0	40

De tweede betekenis van de maximale score wordt eerder per individueel *requirement* bekeken. Dit keer geldt het getal in de kolom 'Score' als de maximale score die kan gegeven aan elke tool. Hoe beter de tool voldoet aan de gevraagde eigenschap, hoe hoger de score zal liggen. 100 geeft aan dat de tool de eigenschap perfect bezit, terwijl 80 nog steeds erg goed is, maar niet perfect. Zodra de scores onder de helft liggen, geeft de kleur oranje aan dat de scores niet meer zo goed zijn. De kleur rood wordt opnieuw gebruikt indien een *requirement* absoluut niet of nauwelijks aanwezig is. Opnieuw reflecteren de scores dit door een getal dichtbij nul toe te kennen.

Hierbij mag niet vergeten worden dat de scores per tool afhangen van de maximale score. Bij de eerste *requirement* zijn zowel 100 als 80 erg goed, terwijl dit overeenkomt met 50 of 40 bij de tweede *requirement*. Onder de 30 of 20 zorgt voor een oranje kleur bij de tweede *requirement* en 10 of 0 zorgt voor een rode inkleuring van de vakjes. Kortom werkt ook deze versie van de tabel met visuele accenten, maar wordt meer diepgang gecreëerd door een score toe te voegen aan de tabel. Deze score zal uiteindelijk beslissend zijn voor het bepalen van de winnaar van de tools.

Voordat er wordt vervolgd naar de laatste vorm van de vergelijkingsmatrix, wordt eerst een optionele tussenstap bondig besproken. Indien mogelijk, kunnen de *requirements* van de Score Matrix geordend worden per categorie. Dit is vooral interessant als er veel *requirements* zijn en bovendien geven categorieën een beter overzicht over het geheel. Verder kan de laatste iteratie van de vergelijkingsmatrix op een veel beknoptere manier weergegeven worden. Dit wordt duidelijk bij de uitleg van de Percentage Matrix.

De **Percentage Matrix** is de laatste iteratie van de vergelijkingsmatrix. Binnen deze tabel worden de scores vergeleken aan de hand van een eindscore, opgebouwd uit percentages die overeenkomen met de eerder gegeven scores. Om een duidelijk voorbeeld te kunnen tonen, worden categorieën toegepast.

Om tot de laatste percentages te komen, worden eerst achterliggend verscheidene berekeningen uitgevoerd. De eerste berekeningen gebeuren per categorie en per tool. Zo wordt aanvankelijk de totale maximale score van een categorie bepaald door de maximale scores binnen de categorie op te tellen. Stel dat een categorie genaamd Categorie 1 drie *requirement*s bezit, met elk een maximale score van 100, 50 en 50. De som hiervan bedraagt 200. 200 is dan de totale maximale score voor Categorie 1.

De gegeven scores voor de *requirement*s binnen Categorie 1 voor een tool zijn natuurlijk niet allemaal perfect. Voor Tool 1 bedragen de scores respectievelijk 80, 50 en 10. Dit telt op tot 140. Voor Categorie 1 scoort Tool 1 dus 140 op 200. Als dit wordt omgezet naar een percentage, wordt het 70%. Dit is niet slecht, maar Tool 3 scoort beter op de onderliggende *requirement*s. Het percentage voor Tool 3 bedraagt 80% voor Categorie 1. Tool 2 scoort ondermaats en behaalt slechts 50%.

Om de werking van de Score Matrix goed te illustreren, wordt een tweede categorie erbij gehaald, namelijk Categorie 2. Hierbij liggen de scores helemaal anders. Tool 1 behaalt 90%, Tool 2 heeft 80% behaald en Tool 3 eindigt met 40%. Hoe wordt nu de uiteindelijke winnaar bepaald?

De beste tool wordt gekozen uit de tool met als totaal het hoogste percentage. Eerst worden de maximale percentages opgeteld. Dit zijn 2 categorieën met elk een maximaal percentage van uiteraard 100%. Het maximale totaal bedraagt dus 200. Vervolgens wordt opnieuw per tool het gegeven percentage opgeteld. Voor Tool 1 is dit 160, voor Tool 2 is het totaal 130 en Tool 3 rondt af met een totaal van 120. Als elke score omgezet wordt naar een percentage, is het eindpercentage van elke tool gekend. Het resultaat van dit voorbeeld is te zien in tabel 3 hieronder.

Tabel 3: Percentage Matrix

	Tool 1	Tool 2	Tool 3
Categorie 1	70%	50%	80%
Categorie 2	90%	80%	40%
TOTAAL	80%	65%	60%

Het eindpercentage vertelt dat Tool 1 de uiteindelijke winnaar is. De kleuren geven nog steeds visueel weer hoe elke tool ervoor staat, maar in dit geval vertellen de percentages meer dan de kleuren. Een opmerking bij de berekeningen is dat kommagetallen verwaarloosd worden.

Dit volbrengt de volledige doorloop van de vergelijkingsmatrix met alle interne iteraties. De scores worden meer en meer verfijnd om uiteindelijk tot een weldoordachte en goed beargumenteerde bevinding te komen. Deze vorm van vergelijken wordt in het verdere verloop van dit onderzoek toegepast op een aantal tools die MBT toepassen.

3.2.2 Requirements

Om de vergelijkingsmatrix van dit onderzoek vorm te geven, zijn twee onderdelen nodig. Als eerste worden de vereiste eigenschappen voor de tools besproken. Deze *requirements* zijn deels door vooronderzoek bepaald en deels door deze te bespreken met Refleqt zelf. Refleqt heeft een aantal suggesties gegeven over welke eigenschappen zeker aan bod moeten komen en heeft ook voorkeuren doorgegeven over welke *requirements* belangrijk, minder belangrijk of *nice-to-haves* zijn voor het bedrijf. Zoals al eerder vermeld, zijn niet alle eigenschappen evenveel van belang als voor andere bedrijven. Vandaar dat de mening van Refleqt van doorslaggevende waarde is. Deze categorisering van de *requirement*s wordt op het einde van dit hoofdstuk voorgesteld.

In totaal zijn 10 *requirements* opgesteld. Elke *requirement* wordt hier kort toegelicht om zo de redenering van elke eigenschap te kunnen begrijpen.

Ten eerste moet er binnen de tool gebruik kunnen gemaakt worden van BPMN. BPMN is namelijk een modelering standaard voor het opstellen van diagrammen, het heeft ISO 19510 meegekregen. Op internationaal niveau wordt BPMN dus erkend als een standaard en wordt het wereldwijd gebruikt door verschillende bedrijven. Het feit dat BPMN zo frequent toegepast wordt door bedrijven, heeft ervoor gezorgd dat meer en meer bedrijven ook compatibel willen zijn met deze standaard. Het is dus geen verrassing dat Refleqt wilt dat BPMN binnen een potentiële MBT-tool verwerkt zit. [17], [20]

De volgende eigenschap bedraagt de mogelijkheid tot rapporten of het geven van een soort eindoverzicht. Deze functionaliteit is ter beschikking bij Cucumber, een testing technologie waar Refleqt veel van gebruik maakt. De rapporten van Cucumber zijn altijd van grote waarde geweest voor Refleqt, dus zou die functionaliteit binnen een MBT-tool een erg groot voordeel vormen.

MBT is in staat testcases te genereren op alle niveaus, maar de generatie kan ook gestuurd worden door een bepaalde aanpak of strategie te kiezen. Zoals al eerder besproken, hangen deze mogelijkheden volledig af van welke tool gebruikt wordt, maar toch blijft het belangrijk om op een heldere manier een duidelijk overzicht te krijgen van welke testcases nu juist gegenereerd zijn. Andere informatie kan ook van belang zijn, zoals op hoeveel tijd de testcases worden gecreëerd of hoeveel procent van het ingegeven model wordt gedekt. Er kan bijvoorbeeld gekozen worden voor volledige coverage van het model, maar dit neemt meer tijd en resources in beslag. Als bijvoorbeeld via bepaalde parameters wordt meegegeven dat slechts een bepaald gedeelte van het model moet worden omgezet naar testcases, dan zal het coverage percentage ook geen 100% bedragen. Kortom, het genereren van testcases kan afhankelijk van de tool volledig naar de behoeften van een tester aangepast worden. Om ervoor te zorgen dat de testcases en de generatie ervan volledig vanzelfsprekend blijven voor het complete team, is een goede rapportfunctionaliteit zeker van belang.

Agile is dé nieuwe manier van werken in de IT-sector. Dit trekt zich door tot in de tools waarmee teams aan de slag willen gaan en bij een MBT-tool is dit niet anders. *Continuous Integration* (CI) vormt dus een centrale eigenschap voor de MBT-tool. Als de tool deze eigenschap bezit, is de tool in staat een deel uit te maken van de CI-cirkel. CI slaat op de samenwerking van verschillende *developer tools* op één pipeline. Door bijvoorbeeld gebruik te maken van Jenkins, een van de meest bekende *automation servers*, is het mogelijk om een hoop taken uit de handen van programmeurs te nemen. Jenkins kan gebruikt worden om diverse taken te automatiseren en deze bijvoorbeeld een aantal keer per dag te laten uitvoeren. Deze taken kunnen deployments zijn, het laten lopen van testen en nog zoveel meer. Jenkins heeft bovendien ontzettend veel plug-inmogelijkheden zodat het integreren van nieuwe tools nooit veel moeite kost. Aangezien Jenkins binnen Refleqt en ook de zusterbedrijven van Refleqt gebruikt

wordt, is de mogelijkheid tot CI toch een zekere noodzaak voor het bedrijf. Hoe meer geïntegreerd kan worden op de pipeline, hoe meer tijd bespaard kan worden en hoe efficiënter kan worden gewerkt. [19]

Aansluitend op CI-mogelijkheden zijn **mogelijkheden tot automatisatie**. Deze eigenschap ligt heel dicht bij CI. CI slaat meer op de integratie binnen een pipeline zodat automatisch een hele reeks aan taken vervuld kan worden. Deze taken komen uit verscheidene tools, maar kunnen dankzij een algemene pipeline met elkaar samenwerken. CI zorgt dus voor samenwerking tussen verschillende tools zodat verschillende tijdrovende taken uit de handen van ontwikkelaars genomen kunnen worden.

Wat is automatisatie precies in tegenstelling tot CI? Automatisatie kan in dit geval via een perspectief binnen de tool zelf bekeken worden, terwijl CI doelt op automatisatie tussen meerdere tools. Het automatiseren van bepaalde taken binnen de tool zelf is ook van enorm belang. Aangezien het eigenlijk de bedoeling is van een MBT-tool dat het genereren van testcases automatisch gebeurt, zal deze eigenschap dus hoogstwaarschijnlijk wel aanwezig zijn bij alle tools. Het verschil tussen de tools zal liggen aan hoe en op welke manier automatisatie wordt toegepast. De grootste factor bij automatisatie is hoeveel tijd er uitgespaard kan worden en dus hoeveel efficiënter een tester of ontwikkelaar kan werken. Hoe goed automatisatie verwerkt is binnen de MBT-tools wordt uitgewezen door de vergelijkingsmatrix.

Een andere factor van een MBT-tool is **snelheid**. Hoe snel is de tool in het algemeen, of hoeveel hebben bepaalde functies nodig? Zoals al eerder wordt aangegeven, dient een MBT-tool om kostbare tijd uit te sparen. Als de generatie van testcases uiteindelijk toch nog meer tijd nodig heeft dat het gewoon manueel uitwerken van de testcases, heeft het ook geen nut om een tool ervoor aan te schaffen. Tijd is geld en in de business wordt steeds meer en meer geëist van het hele team. Er moet steeds meer gedaan worden op minder tijd.

Een tool kan fantastische functionaliteiten bevatten, maar als er geen overzichtelijke **documentatie** ter beschikking is, valt het dikwijls absoluut niet mee om op korte tijd de tool in te brengen in het proces binnen een bedrijf. Dit kost opnieuw tijd en geld en geeft bovendien geen aangenaam welkom aan degene die de tool wilt integreren binnen een project. De precieze installatie, de werking van de tool, eventuele tips of een goede tutorial; er is zoveel dat de ervaring van de installateur zoveel prettiger zou kunnen maken.

De **uitvoering** van een MBT-tool is ook van belang. Hoe uitgebreid zijn de opties van de tool? Kan de tool ook op de achtergrond gebruikt worden, of enkel op de voorgrond? Aangezien deze eigenschap nogal algemeen en niet-specifiek is, kan de uiteindelijke score van de verschillende tools ofwel erg dicht bij elkaar liggen, of erg ver uit elkaar. Dit zal uit onderzoek moeten blijken.

Ook het **aanpassingsvermogen of de uitbreidbaarheid** van een tool mogen niet ontbreken, zeker naar de toekomst toe. De IT-sector blijft altijd groeien, soms zelfs sneller dan verwacht. Hierop anticiperen is een uitdaging, maar mag zeker niet naar de achtergrond geschoven worden. Eveneens worden de limieten van een tool in rekening gebracht. Verder sluit deze eigenschap opnieuw aan bij andere *requirement*s, namelijk of de tool ook kan samenwerken met andere tools of technologieën.

Een van de laatste *requirement*s bedraagt de mogelijkheden op vlak van **programmeertalen**. Sommige tools zijn in staat de gegenereerde testcases meteen om te zetten naar bruikbare testen in bepaalde testing technologieën zoals Selenium, terwijl andere tools het houden op de creatie van de testcases. Hoe dat de testcases uiteindelijk omgezet kunnen worden naar bruikbare testen, is ook een facet van deze *requirement*.

Tot slot is er nog een *requirement* over, namelijk de **prijs**. Een score toekennen aan deze *requirement* lijkt in theorie niet moeilijk; de duurste tool krijgt de laagste score, de goedkoopste de hoogste. In grote lijnen klopt dit, maar door abonneesystemen en andere manieren om gebruik te kunnen maken van de tool, bemoeilijken de bepaling van welke prijs de betere is. Verder hangt de waarde van de prijs ook af van wat een gebruiker krijgt voor die prijs. Aangezien een goede prijs dikwijls niet meer afhangt van welke de goedkoopste is, moeten dus meerdere factoren in acht worden gehouden. Een score bepalen voor elke tool vormt dus zeker een uitdaging.

Om dit hoofdstuk correct af te ronden, mag de categorisering van de tools alleszins niet vergeten worden. Na een korte bespreking met Refleqt zijn er drie categorieën bepaald, waarbij elke categorie weergeeft hoe cruciaal de *requirement*s binnen de onderverdeling zijn. De *requirement*s die absoluut in de tools moeten zitten, behoren tot de categorie genaamd Cruciaal. Eigenschappen die belangrijk zijn maar niet kritiek, maken deel uit van de categorie Medium. De laatste categorie heet Optioneel en bestaat uit leuke extraatjes. Deze *requirement*s zijn helemaal niet vereist binnen de tools, maar kunnen een toegevoegde waarde vormen. Hieronder wordt in tabel 4 een overzicht gegeven van de categorieën samen met de bijbehorende *requirement*s.

Tabel 4: Overzicht categorieën vergelijkingsmatrix en bijbehorende requirements

Categorie	Requirement
Cruciaal	BPMN
	Rapport functionaliteit
	CI-mogelijkheden
	Automatisatiemogelijkheden
Medium	Aanpassingsvermogen of uitbreidbaarheid
	(Programmeer)talen of technologieën
	Prijs
Optioneel	Snelheid
	Documentatie
	Uitvoering

3.2.3 Tools

De vergelijkingsmatrix bevat de vereiste requirements, maar heeft natuurlijk ook tools nodig om te kunnen vergelijken. Verschillende tools passen MBT toe. Drie tools zijn geselecteerd op basis van een eerste indruk en of ze twee eigenschappen bezitten, namelijk of BPMN een mogelijkheid is en of er voldoende opties zijn qua gebruik van de tool. Voor dit onderzoek zijn drie tools geselecteerd: Conformiq Creator, MBTsuite en Yest.

Conformiq Creator is een MBT-tool van Conformiq en is in staat functionele test processen te automatiseren en de SDLC te versnellen. Deze tool is daartoe in staat door gebruik te maken van MBT. Conformiq zelf is een bedrijf dat ontstaan is in 1998. Het bedrijf is actief op internationaal niveau; zowel in de Verenigde Staten, Finland, Zweden, Frankrijk en India. Verder is Conformiq een partner met alle grote SDLC-toolproviders, zoals Atlassian, Jenkins, Git, Microsoft, SoapUI, Selenium en meer. Het doel van het bedrijf is om zo veel mogelijk test automatisatie tot bij de klant te brengen en om de hoeveelheid onderhoud tot een absoluut minimum te beperken. [21]

Figuur 13: Logo Conformiq Creator [29]

De tweede MBT-tool heet **MBTsuite**. Ook bij deze tool staat automatisering van het genereren van testcases centraal. Het bedrijf achter MBTsuite heet sepp.med, het werd in 1980 opgericht door Franz-Josef Prester en is gevestigd in Röttenbach, Duitsland. Het bedrijf is gespecialiseerd op het gebied van medische technologie, maar ook in de farmaceutische industrie, *automotive engineering* en *avionics*. Om op de tool terug te komen, is het centrale thema het verlagen van kosten in het testproces en tegelijkertijd het verhogen van de kwaliteit. [22], [24], [25]

Figuur 14: Logo MBTsuite [30]

De laatste MBT-tool is van het bedrijf Smartesting, namelijk **Yest**. YEST profileert zich als een visuele test design solution dat MBT toepast. Het bedrijf achter deze tool, Smartesting, is gevestigd in Frankrijk en heeft als focus een grafische manier van testen, met als specialisatie MBT. Yest werd gelanceerd in 2017. De bedoeling van Yest is om professionals een tool aan te bieden dat zowel visueel als agile het proces van functionele validatie van IT-systemen sterk vergemakkelijkt. [23]

Figuur 15: Logo Yest [23]

3.2.4 De vergelijking

Alle nodige elementen voor de vergelijkingsmatrix zijn samengebracht. Om de daadwerkelijke vergelijking te beginnen, wordt gestart met de aanwezigheid van de verschillende requirements te bespreken bij elke tool. Conformiq Creator bijt de spits af.

Alle informatie in de onderstaande beschrijvingen en matrixen binnen dit hoofdstuk is terug te vinden op de webpagina's van de MBT-tools. Een deel van de weergegeven informatie is terug te vinden op andere pagina's van dezelfde websites. Deze pagina's van de websites zijn niet als aparte bron aangegeven. [21], [22], [23], [24], [25]

3.2.4.1 Conformiq Creator

Op vlak van BPMN is Conformiq Creator in staat deze te importeren in onder andere de vorm van Visio-modellen. Bij het importeren worden de modellen omgezet naar geoptimaliseerde Creator modellen die binnen Creator zelf gebruikt worden voor de generatie van testcases. [21]

Rapport functionaliteit is verwerkt binnen Creator. Na de generatie van testcases is voornamelijk een grafisch overzicht ter beschikking met een opsomming van informatie zoals test coverage, traceerbaarheid en *message sequence charts*. Dankzij bijvoorbeeld animaties kan altijd perfect waargenomen worden hoe een bepaalde testcase wordt uitgevoerd, of in welke volgorde. Verder is Creator ook in staat problemen in de generatie van testcases te detecteren met behulp van een model browser en zorgt een *AI-based test engine* voor uitvoerbare test scripts, test input data, testcase namen en test step documentatie. Over het genereren van aparte rapporten wordt niets specifiek vermeld, maar de integratie met tools zoals Jira geven aan dat traceerbaarheid zeker een optie is. [21]

Op vlak van CI zijn er mogelijkheden genoeg. Conformiq zegt partner te zijn van verschillende grote softwarebedrijven, zoals Atlassian Jira, Jenkins en meer. Hierbij hoort niet alleen het uploaden en traceren van geautomatiseerde testcases binnen ALM-tools, een andere tool van Conformiq zet zelfs een stap verder qua CI. Conformiq Transformer is een andere tool dat zich voornamelijk focust op het automatisch uitvoeren van testcases. Met andere woorden, Transformer sluit perfect aan op de werking van Creator. Bovendien biedt Conformiq een pakket aan genaamd Conformiq 360° Test Automation waarin beide tools zitten, zodat integratie van beide tools des te makkelijker wordt. [21]

Aansluitend op CI zijn mogelijkheden in automatisatie. End-to-end automatisatie behoort tot de eigenschappen van Creator, alsook het omzetten van gebruiker interactie naar test scripts. Bijkomend is validatie voor geautomatiseerde executie. Op die manier kan de tussenkomst van een effectieve persoon tot een minimum beperkt worden. [21]

Bij Creator zijn er ook extra mogelijkheden tot uitbreiding of aanpasbaarheid. Zo kan ook het uitvoeren van de gegenereerde testcases geautomatiseerd worden door gebruik te maken van een extra tool, namelijk Conformiq Transformer. Aangezien dit een aparte tool is, zullen hieraan wel extra kosten verbonden zijn. Zoals al eerder vermeld, kunnen beide tools ook samen aangekocht worden door Conformiq 360° Test Automation aan te schaffen. Ook deze optie zal uiteindelijk duurder zijn dan Creator op zich. [21]

Development en test executie staan nog steeds centraal in een IT-project, dus spelen de (programmeer)talen en technologieën een belangrijke rol. Volgens Conformiq worden test scripts gegenereerd in elke taal en in elk format. Dit zorgt voor naadloze integratie met *verschillende test execution frameworks*; test documentatie en test plannen worden in Excel of in een ALM-tool gegenereerd. Verder worden ook niet-Engelse modellen en rapportage ondersteund en kunnen stakeholders *customized* rapporten bekijken in Excel, HTML en XML. [21]

De prijs van Conformiq Creator is volledig op aanvraag. De tool kan apart aangekocht worden, maar ook in combinatie met andere tools zoals het Conformiq 360° Test Automation pakket. [21]

De **snelheid** van Conformiq Creator speelt ook een rol. Helaas zijn hier geen details over te vinden. Wel zijn er veel beloften over een sterke verlaging in de time-to-market, aangezien de software ontworpen is met hoge herbruikbaarheid en een eenvoudige manier om aanpassingen te maken. [21]

De beschikbare **documentatie** en hoe instapklaar een tool is, mogen ook niet vergeten worden. Bij Conformiq Creator zijn er onder andere online courses te verkrijgen op aanvraag, maar ook een volledig YouTube-kanaal met trainingsvideo's en uitleg over nieuwe releases zijn beschikbaar. Verder zijn ook *data sheets* en *white papers* te vinden op de website en wordt bovendien een Quick Start pakket aangeboden. Hierin zit extra ondersteuning qua installatie en integraties, maar ook knowledge transfers maken hier deel van uit. Extra informatie over andere tools zoals Transformer kunnen ook aangevraagd worden. [21]

Qua **uitvoering** van de tool en extra opties springt een feature in het oog, de *AI-based reverse engineering* technologie. Bij BPMN wordt deze feature ook al besproken, maar ook bij dit onderdeel komt deze feature naar voren. De mogelijkheid tot het importeren van verschillende soorten testcases en modellen is erg handig, zoals *Gherkin feature files*, Visio-modellen, BPT, XSD en WSDL structuren, UFT/LeanFT, Selenium tests en *data object repositories* en nog meer. [21]

Verdere mogelijkheden van de tool zijn te zien in de YouTube-video's van Conformiq. In deze video's wordt met de tool zelf gewerkt en is te zien dat er een overdaad aan opties en keuzes ter beschikking is. [21], [26], [27]

3.2.4.2 MBTsuite

Op vlak van **BPMN**, is de documentatie van MBTsuite nogal verwarrend. Langs de ene kant wordt op de website en in de YouTube-video aangegeven dat voor *modeling* gebruik gemaakt kan worden van onder andere Visio, een tool dat zich profileert met de BPMN-standaard. Echter, bij het uitproberen van een free trial staat in de bijgeleverde documentatie aangegeven dat MBTsuite zich focust op UML. Bij nader onderzoek van de meegeleverde user guide kan ervan uit gegaan worden dat BPMN een optie is binnen de tool, aangezien *activity diagrams* behoren tot de mogelijke modellen. Vermoedelijk is de guide ook verouderd, aangezien in het document over Visio niets terug te vinden is, terwijl andere *modeling* tools wel vernoemd worden. Op de website en in de video daarentegen wordt Visio wel vernoemd. Deze inconsistentie kan bij een introductie bij een potentiële klant voor verwarring zorgen. [22], [24], [28], [31]

Het genereren van **rapporten** lijkt op een andere manier te werken binnen MBTsuite. Binnen het programma zelf is een apart tabblad voorzien genaamd *Statistics*. *Statistics* geeft zoals de naam al vermoed, een overzicht weer van verschillende statistieken, zoals op hoeveel tijd een testcase tree werd gegenereerd, hoeveel procent van het ingevoerde model werd omgezet naar testcases, van welke strategie voor de generatie van testcases werd gebruik gemaakt en nog veel meer. In de instellingen van MBTsuite kan ook ingesteld worden welke velden al dan niet zichtbaar zijn in het overzicht en op welke volgorde deze staan. *Statistics* geeft dus een correct overzicht weer met een overmaat aan beschikbaar cijfermateriaal, maar het visuele aspect zoals bij Cucumber rapporten ontbreekt. In een oogopslag een idee hebben van de generatie van testcases, is dus geen mogelijkheid. Verder lijkt het Statistics tabblad statisch te zijn. De mogelijkheid tot het eventueel exporteren van een Statistics rapport is op het eerste zicht niet mogelijk. [22], [31]

Integreren met CI-tools is mogelijk, maar in zeer beperkte mate. Zo is er geen sprake van samenwerking met grote, bekende spelers zoals Jenkins of Jira. Hoewel Jenkins erom bekend staat integratiemogelijkheden te bieden voor bijna elke tool op de markt, zou het niet hebben van een geïntegreerde samenwerking een tester heel wat kopzorgen bezorgen. Wel kan er onder andere met HP Application Lifecycle Management of IBM Rational gewerkt worden, maar verder is er specifiek op vlak van CI-mogelijkheden niet veel informatie terug te vinden. [22], [31]

Hoewel de centrale gedachte van MBT het automatisch genereren van testcases bedraagt, is van automatiseringsmogelijkheden binnen MBTsuite niet veel sprake. Na het doornemen van de gebruikershandleiding blijft de eerste indruk van toch nog veel te moeten doen toch wel hangen. Het bepalen van de exacte eigenschappen voor een zeer specifieke generatie van testcases is zeker mogelijk binnen deze tool, maar de tijd om alles in te stellen, mag zeker niet onderschat worden. Dit zorgt voor vragen of deze tijd uiteindelijk ook teruggewonnen kan worden. [22], [31]

Het toekomstperspectief van MBTsuite ziet er twijfelachtig uit. Deze tool werkt goed binnen een bedrijf dat niet sterk afhankelijk is van het CI-principe of dat werkt op een agile manier, maar is dat wel een echt voordeel? Agile wordt gezien als de laatste nieuwe evolutie op vlak van een manier van werken binnen IT, dus zijn tools of bedrijven die deze wijze niet toepassen dan wel *future-proof*? Langs de andere kant is MBTsuite perfect in staat deze aanpassingen door te voeren om toch nog mee te zijn met de agile methodiek. Verder zou een modernisering van de tool geen slechte investering zijn, aangezien er echt wel ruimte is voor een uitbreiding van de al bestaande opties. [22], [31]

Volgens de gebruikershandleiding kunnen de gegenereerde testcases geëxporteerd worden in de volgende programmeertalen of tools: CSV, Excel, Word, HTML, Java, C en Python. Op de website staan nog een aantal extra tools, namelijk Microsoft .net en Visual Studio. [22], [31]

Qua prijs zijn er opnieuw geen details terug te vinden. Hiervoor moet er contact gemaakt worden met het bedrijf zelf. [22]

Ook op vlak van snelheid is er ook niet bekend. Zoals bij Conformiq worden beloftes gemaakt dat het ten opzichte van het schrijven van manuele testcases veel tijd zou besparen, maar dat is net het doel van MBT. Cijfermateriaal om mee te vergelijken, is dus niet beschikbaar. [22], [31]

De bijgeleverde documentatie van de *free trial versie* van MBTsuite, kan voor personen met een laag niveau van testing kennis een behoorlijke uitdaging vormen. Hoewel in het begin van de gebruikershandleiding de afkortingen worden toegelicht, blijkt de rest van de handleiding in erg technisch jargon geschreven. Verschillende termen worden zelden of niet uitgelegd, wat zorgt voor een zoektocht binnen een gebruikershandleiding, iets dat eigenlijk niet de bedoeling mag zijn. Verder lijkt de documentatie niet volledig up-to-date te zijn en zijn verbeteringen van een vorige versie van de handleiding nog volledig zichtbaar. Als eerste indruk voelt dit bijzonder slordig en ondoordacht. [22], [31]

Binnen MBTsuite zijn er qua uitvoering opties en keuzes aan overdaad. Kiezen welke strategie op een model wordt uitgevoerd, kiezen welke velden zichtbaar zijn of niet en nog veel meer kan allemaal beslist worden door de gebruiker. Langs de andere kant is de terminologie van een aantal functies niet helemaal duidelijk. Zo is een volledig hoofdstuk binnen de gebruikershandleiding besteed aan ‘Delta generatie’. Waarschijnlijk wordt hiermee eenvoudigweg de generatie van de testcases bedoeld, maar wat heeft het woord delta er dan mee te maken? Zelfs na het doornemen van dit hoofdstuk, is dat nog steeds niet 100% duidelijk. [22], [31]

3.2.4.3 Yest

Over BPMN wordt niet nadrukkelijk gesproken. Qua mogelijkheden van modellen is er ook niet veel informatie te vinden, maar de werking van de *modeleditor* van Yest zelf wordt wel toegelicht. Een voorbeeld van de symbolen van deze editor is in figuur 16 te zien. Deze editor maakt op het eerste zicht gebruik van de herkenbare principes van een *activity diagram*, zo wordt ook hier een keuzeruit gebruikt. Maar daar blijft het bij. Mogelijkheden om externe modellen te importeren zijn niet terug te vinden, wat toch wel een teleurstelling vormt. In tegenstelling tot de verscheidene mogelijkheden met de uitkomst van deze MBT-tool voor CI, is er een schril contrast met de opties voor het inbrengen van modellen. [23]

Figuur 16: Yest: grafische elementen van de modeleditor

Rapport functionaliteit zoals bij Cucumber is niet helemaal hetzelfde bij Yest. Het genereren van rapporten wordt niet effectief aangehaald binnen de informatie van Yest, maar er wordt wel aangegeven dat er integraties mogelijk zijn met tools die dit eigenlijk al doen, zoals TestRail. Verder worden wel veel tussentijdse, visuele overzichten aangeboden, zoals vlak voor het genereren van testcases. Instellingen en dergelijke kunnen zo voor een laatste keer gecontroleerd en eventueel gecorrigeerd worden. [23]

Smartesting beseft dat CI-mogelijkheid een belangrijke rol speelt voor nieuwe klanten. Integratie met test management tools zoals ALM, Xray, TestRail, TestLink en meer behoort tot de mogelijkheden van Yest, maar ook Jira, Cucumber en Selenium worden vernoemd als mogelijke integraties. Yest speelt in op *end-to-end* automatisering met aandacht voor agile principes en visuele presentatie van allerlei gegevens. Zo wordt in de documentatie van Yest aangegeven dat een verbinding met Jira met slechts enkele kliks gemaakt kan worden. Tot slot bestaat er ook de mogelijkheid tot integratie met versiebeheersystemen zoals Git of SVN. [23]

Aan automatisatie binnen Yest zelf wordt ook niet ondergedaan. Zo wordt er gebruik gemaakt van implementatie van '*good practice keywords*', deze sleutelwoorden maken het waarschijnlijk makkelijker om regelmatig gebruik te maken van dezelfde variabelen, gegevens of functies binnen Yest. Verder herkent Yest het verschil tussen de context van de verschillende sleutelwoorden, zoals op vlak van business, data of technische logica. Daarnaast kunnen scripts gepubliceerd worden in de meeste market tools. Voor de publicatie in de geautomatiseerde omgeving van zo een script, wordt eerst alle informatie weergegeven in een tabblad, inclusief de initialisatie, afronding en verdere instellingen van het script. Een laatste opmerking bij automatisatie is dat binnen de *modeleditor* en op mogelijk andere plaatsen

suggesties worden gegeven voor het vervolledigen van scenario's of dergelijke. Afhankelijk van hoe effectief deze automatische aanvulling werkt, kan deze erg nuttig zijn of juist voor contraproductiviteit zorgen. [23]

Het aanpassingsvermogen of de uitbreidbaarheid van Yest bevat veel potentieel. Verbindingen met CI-tools voor de DevOps cyclus zijn al een vaste waarde binnen Yest, maar de tool kan hier nog veel verder in gaan. Zo zouden integratie mogelijkheden met *modeling tools* een mooie uitbreiding kunnen vormen.

Met welke (Programmeer)talen of technologieën Yest compatibel is, wordt nergens expliciet aangehaald, buiten Cucumber en Selenium. Verder wordt aangehaald dat de focus ligt op de principes van ATDD en BDD. Het genereren van manuele testcases wordt wel naar voren gebracht, maar dit is niet zo interessant voor onder andere Refleqt. [23]

Over de prijs wordt opnieuw niets prijsgegeven. Om de tool gewoon eens te kunnen uitproberen, dient er een afspraak gemaakt te worden om zo verdere details te verkrijgen. [23]

Yest is de eerste tool die concrete cijfers meegeeft op vlak van snelheid. Allereerst zou een volledig nieuwe gebruiker in staat moeten zijn te werken met Yest binnen twee uren. Verder zou de productiviteit van het volledige test proces toenemen met 20%. Dit percentage wordt onderverdeeld in bepaalde aspecten binnen het test proces. In de design fase, dus het opstellen van de nodige modellen, groeit de productiviteit 20 tot 30%. Onderhoudstesten, of testen die dikwijls uitgevoerd worden om de algemene werking van software te controleren, zorgen voor een groei van 40 tot 50% op vlak van productiviteit. Dit komt doordat het maken van aanpassingen binnen bestaande modellen of testen erg vlot verloopt. Verder is er een veel beter begrip van de tool zelf, met als gevolg dat er 50% minder misverstanden voorkomen. Dit zorgt op zijn beurt voor meer controle over test coverage en traceerbaarheid. [23]

De website van Smartesting bevat een verzameling van *data sheets*, webinars, *white papers* en tutorials. De *data sheet* over Yest geeft in enkele pagina's weer wat een potentiële gebruiker kan verwachten van het programma. De webinars zijn terug te vinden op YouTube en bespreken elk een specifiek topic rond de automatisering van testing, met als focus de tool Yest. Verder bespreekt de *white paper* de uitdagingen van een steeds meer gedigitaliseerde werking binnen een organisatie, en hoe dat Yest hier uiteindelijk bij komt kijken. Al deze documentatie komt dus zeker van pas, alleen is de meeste documentatie in het Frans te verkrijgen, of gedeeltelijk in het Engels. Dit is tegenwoordig geen groot obstakel meer dankzij verschillende (gratis) vertaaltools die online te vinden zijn, maar het blijft een extra, onnodige stap. Zeker om te kunnen concurreren op internationaal niveau, is het ter beschikking stellen van middelen in het Engels een absolute must. Engels blijft in het digitale landschap een taal met een enorm bereik. [23]

Qua uitvoering wordt binnen Yest voornamelijk gesteund op drie pilaren, namelijk de agile methodiek met CI, overal aanpassingen kunnen doorvoeren en alles op een visueel aangename manier presenteren. Smartesting beseft het belang van agile werken en zet dus hard in op het inbrengen van Yest binnen de DevOps cyclus. Verder wordt veel aandacht besteed aan het gemak van de gebruiker, met name dat op bijna elk moment een aanpassing kan worden doorgevoerd. De rest van de informatie wordt aangepast zodat de gebruiker geen extra werk meer heeft. Tot slot werkt Yest erg visueel. Aangezien een beeld meer dan duizend woorden kan vertellen, is dit een niet te onderschatten voordeel. Een mens kan op die manier veel sneller (nieuwe) informatie verwerken en zich focussen op de informatie die hij op dat moment nodig heeft, in plaats van te zoeken naar de nodige informatie. Alle gewonnen tijd is winst.

3.2.4.4 Requirements Matrix

Alle invullingen van de requirements zijn hieronder kort samengevat samengebracht in de Requirements Matrix. Zoals in hoofdstuk 3.2.1 wordt aangegeven, geven de drie kleuren een eerste beeld van hoe goed een MBT-tool ervoor staat. In de volgende matrixen wordt dit beeld verfijnd.

Tabel 5: Requirements Matrix toegepast

	Conformiq Creator	MBTsuite	Yest
BPMN	Kan BPMN & Visio importeren en omzetten tot bruikbare modellen.	Inconsistente documentatie, maar Visio is wel beschikbaar. Focus ligt op UML, maar bv. *activity diagrams* zijn een optie.	Enkel werken met de modeleditor binnen de tool zelf, geen info over mogelijkheden tot importeren van andere tools.
Rapport functionaliteit	Model browser detecteert problemen, *AI-based test engine* genereert uitvoerbare test scripts, test input data, testcase namen en test step documentatie. Generatie traceerbaarheid, *message sequence charts* en grafisch overzicht test coverage.	Statistics tabblad: geeft een overzicht van tijd, coverage percentage en nog veel meer, maar mist visueel aspect en mogelijkheid tot inbrengen binnen een CI-cyclus.	Integratie mogelijk met tools die rapporten kunnen aanmaken, zoals TestRail. Tussentijdse, visuele overzichten van instellingen, data sets en meer.
CI-mogelijkheden	Integraties met Atlassian Jira, Micro Focus ALM, CA Rally, IBM DOORS, etc. Uploaden en tracken van geautomatiseerde testcases in ALM-tools, Conformiq Transformer.	In beperkte mate: HP Application Lifecycle Management, IBM Rational, maar geen grote spelers zoals Jenkins of Jira.	Integratie met ALM, Xray, TestRail, TestLink en meer. Technologieën: Jira, Cucumber en Selenium. Versiebeheersystemen zoals Git of SVN.
Automatisatie mogelijkheden	End-to-end automatisatie, gebruiker interactie kan in test scripts omgezet worden met validatie voor geautomatiseerde executie.	Buiten de generatie van testcases zelf, niet veel automatiseringsopties.	End-to-end automatisatie, slimme *keywords* die de efficiëntie bevorderen, suggestie functionaliteit voor aanvullen van scenario's, etc.
Aanpassingsvermogen of uitbreidbaarheid	Conformiq Transformer: geautomatiseerd uitvoeren van testcases. Conformiq 360° Test Automation: bevat zowel Creator als Transformer.	Tool is niet echt *future-proof*, maar bevat zeker de capaciteit om een inhaalbeweging te maken.	CI-mogelijkheden staan al redelijk op punt, maar kan zeker nog verbeteren op vlak van *modeling tools*.
(Programmeer)talen of technologieën	Uitvoerbare test scripts: generatie in elke taal, in elk format. Integratie met *test execution frameworks*; test documentatie en test plannen worden in Excel of ALM-tool gegenereerd. Ondersteuning niet-Engelse modellen en rapportage.	CSV, Excel, Word, HTML, Java, C, Python, Microsoft .net en Visual Studio.	Cucumber, Selenium. De aanpak van ATDD en BDD worden sterk toegepast. Testcases voor manueel te testen vormen ook pilaar van Yest.

Prijs	Volledig op aanvraag. Aparte aankoop van tools mogelijk, pakketten zijn ook een optie.	Niets teruggevonden. Volledig op aanvraag.	Op aanvraag.
Snelheid	Geen details teruggevonden. Wel beloften dat *time-to-market* sterk wordt verlaagd door hoge herbruikbaarheid eenvoudige manier om aanpassingen te maken.	Geen specifieke details; "het bespaart tijd ten opzichte van het manueel uitschrijven van testcases".	Algemene groei in productiviteit: 20%.
Documentatie	Online courses op aanvraag; YouTube-kanaal met trainingsvideo's en uitleg over nieuwe releases; *Data sheets*; *White papers*; Quick Start pakket: extra ondersteuning met *knowledgde transfers,* installatie, integratie, etc.	YouTube-video's beschikbaar, weliswaar in het Duits. Handleiding redelijk technisch en niet up-to-date. Dikwijls verwarrende informatie in verschillende bronnen (website, handleiding).	*Data sheets*, webinars, *white papers*, tutorials beschikbaar, weliswaar in het Frans. Wel eenvoudig te vertalen via een online tool.
Uitvoering	*AI-based reverse engineering* technologie: importeren van manuele testcases, Gherkin feature files, Visio, BPT, XSD en WSDL structuren, UFT/LeanFT, Selenium tests en data object repositories, etc. Werking Creator wordt toegelicht in YouTube-video's.	Erg veel keuzemogelijkheden binnen de tool. Terminologie is soms niet duidelijk zoals namen van bepaalde functies, etc.	Drie belangrijke pilaren: agile werken met CI, overal aanpassingen kunnen doorvoeren en hoofdzakelijk visueel informatie weergeven.

3.2.4.5 Score Matrix

De volgende stap is een effectieve score toekennen aan elke requirement van elke tool. De maximale scores zijn bepaald aan de hand van de categorieën, aangezien elke categorie een niveau van belangrijkheid voorstelt. De kleuren worden opnieuw gebruikt om visueel een stand van zaken weer te geven.

Tabel 6: Score Matrix met categorieën toegepast

	Score	Conformiq Creator	MBTsuite	Yest
CRUCIAAL				
BPMN	100	100	60	20
Rapport functionaliteit	100	80	50	50
CI-mogelijkheden	100	90	50	100
Automatisatie mogelijkheden	100	90	10	100
MEDIUM				
Aanpassingsvermogen of uitbreidbaarheid	50	40	10	30
(Programmeer)talen of technologieën	50	50	40	30
Prijs	50	20	20	20
OPTIONEEL				
Snelheid	20	5	5	15
Documentatie	20	20	10	15
Uitvoering	20	15	10	15

3.2.4.6 Percentage Matrix

Uiteindelijk wordt de laatste matrix opgesteld, de percentage matrix. Bij de vorige matrixen is het misschien al duidelijk, maar het vermoeden dat Conformiq Creator als winnaar uit de hoed zou komen, is bij deze bevestigd. Dit is geen verrassing, op bijna alle vlakken steekt Conformiq Creator er met kop en schouders bovenuit.

Tabel 7: Percentage Matrix toegepast

	Conformiq Creator	MBTsuite	Yest
CRUCIAAL	90%	42%	67%
MEDIUM	73%	46%	53%
OPTIONEEL	66%	41%	75%
TOTAAL	76%	43%	65%

3.3 Experiment

Oorspronkelijk vormde het experiment de laatste fase van dit onderzoek. Het was de bedoeling de beste tool toe te passen op een van de diagrammen van de stageopdracht, namelijk de Crap App. Helaas bleek tijdens het onderzoek van de tools, dat slechts een tool de mogelijkheid aanbood tot het uitproberen van de tool. Dit bleek MBTsuite te zijn. Om toch nog een praktisch gedeelte toe te voegen aan dit onderzoek, werd besloten met deze tool een toepassing uit te voeren.

Hierbij kwam het volgende obstakel aan bod, namelijk dat de *free trial* die werd aangeboden door MBTsuite, gewoonweg niet werkte. Verschillende pogingen werden ondernomen om toch nog iets te kunnen realiseren, maar het mocht niet baten. Vandaar dat er dus geen experiment gedeelte hoort bij dit onderzoek.

4 Besluit

4.1 Bespreking onderzoeksresultaten

Om een goed besluit te vormen, wordt teruggekeken naar het begin van het onderzoek. De onderzoeksvragen vormen namelijk de basis en motivatie van het volledige onderzoek. Allereerst werd een antwoord gezocht op hoe MBT precies werkt en wat het concept betekent voor de testing industrie. Verder werd gekeken naar de potentiële waarde voor Refleqt, dit wordt toegelicht in 4.2 Aanbevelingen. Ook tools die gebruik maken van MBT werden onderzocht en tot slot werd een praktisch gedeelte van de tools gevraagd. Bij het onderzoek zijn boeiende antwoorden ontdekt, maar niet alle vragen konden beantwoord worden.

MBT staat voor Model Based-Testing. Het principe van MBT berust op het automatisch genereren van testcases op basis van modellen. Deze modellen stellen het gedrag voor van het te testen systeem, genaamd de *system under test* of SUT. Modellen kunnen op verschillende manieren opgesteld worden; door de transities van de staat van het systeem weer te geven, door beslissingen in functies weer te geven, enzovoort. UML en BPMN zijn beide modeleringstandaarden die op internationaal niveau een standaard bepalen over hoe zo een model opgesteld kan worden. Doordat ze als standaard geaccepteerd zijn, kunnen bedrijven efficiënter met elkaar samenwerken. Het genereren van de testcases kan daarnaast op verschillende manieren aangepakt worden. Deze zogenaamde benaderingen of strategieën bieden de tester een ruime variatie aan mogelijkheden voor de exacte testcases te genereren die de tester nodig heeft. Aangezien er zoveel opties zijn, zorgt dit mogelijk voor verwarring. Vandaar dat bedrijven die een MBT-tool aanbieden, zelf bepaalde strategieën aanbieden binnen de tool. De tester kan zo voor een MBT-tool kiezen met een bijpassende strategie.

Bij een vergelijking van de gevonden definitie met twee externe goede bronnen zijn weinig tegenstrijdigheden gevonden. Bij een bron werd niet meteen besloten dat het model van de SUT het gedrag van een systeem moet weergeven. De bron keek eerst naar systemen op erg laag niveau, bijvoorbeeld individuele software modules. Volgens de bron zou een model van deze modules enkel de input data nodig hebben. Hierna keek de bron naar de evolutie van software systemen en werd besloten dat grotere en complexere systemen een model zouden voortbrengen dat het gedrag van zo een systeem weergeeft. De reden dat deze bron naar deze evolutie keek, was omdat de bron uit 1999 afkomstig is. Daaruit kan geconcludeerd worden dat de technologie van toen niet te vergelijken is met de technologie van nu. Het idee achter MBT is dus ook geëvolueerd.

Een andere kleine tegenstrijdigheid is het feit dat de andere bron de executie van de gegenereerde testcases opneemt in de definitie van MBT. Als de informatie van het onderzoek naar de verschillende MBT-tools erbij genomen wordt, valt al snel op dat de effectieve uitvoering van de testen geen intrinsiek deel uitmaakt van het centrale idee van MBT. Sommige tools voorzien dit als extra mogelijkheid, maar het wordt niet als een vast onderdeel van MBT beschouwd.

Er zijn dozijnen MBT-tools op de markt. Om een eerste onderscheid te maken, is naar drie eigenschappen gekeken, namelijk of de tool tot zekere mate gebruik maakt van BPMN. Verder is het belangrijk om te zien of er voldoende informatie over de tool terug te vinden is en daarna is gecontroleerd of de tool op visueel vlak werkt, niet enkel op bijvoorbeeld *command line*. Deze eigenschappen vormen een eerste filter. Hieruit zijn drie tools geselecteerd: Conformiq Creator, MBTsuite en Yest. Om een winnaar te kunnen bepalen, geeft Refleqt een aantal vereisten aan. Uiteindelijk sluit Conformiq Creator het beste aan bij deze requirements. Normaal zou hierop een praktisch gedeelte toegepast worden, maar omdat zowel de winnaar als de andere tools geen manier aanbieden om de tools eens uit te proberen, moest het experiment gedeelte geschrapt worden.

4.2 Aanbevelingen

Voor Refleqt wordt uiteraard de winnaar van de vergelijking van de tools aanbevolen als de te gebruiken MBT-tool. Eerst en vooral kijkt Refleqt best of het gebruik maken van zo een tool effectief de efficiëntie van het bedrijf vooruit zou helpen. Conformiq Creator is namelijk een vrij grote tool met heel wat uitbreidingsmogelijkheden. Het is perfect mogelijk dat Refleqt geen behoefte heeft aan een (groot) deel van de opties die Conformiq Creator aanbiedt. In dat geval is het niet opportuun voor Refleqt om de tool aan te schaffen.

Langs de andere kant zijn er nog tientallen andere tools op de markt. Sommigen zijn veel kleiner en specialiseren zich op bepaalde technologieën. Andere tools automatiseren slechts bepaalde stukjes van het testproces. Mogelijk werkt dit beter voor Refleqt in plaats van meteen een gigantische tool in gebruik te nemen. Bij dit onderzoek lag de focus op tools die BPMN op de een of andere manier toepassen of gebruiken. Er werd ook vergeleken op andere vlakken, mogelijk kan Refleqt hieruit nieuwe informatie over MBT leren die van nut kunnen zijn voor het selecteren van een goede MBT-tool. Aangezien dus een selectief gedeelte van de MBT-tools onderzocht werd, zijn er nog ontzettend veel mogelijkheden die betere of andere voordelen voor Refleqt bieden. Hieruit vloeien mogelijkheden voor eventuele toekomstige onderzoeksvragen.

Welke tools op basis van opensourcesoftware zijn er en wat bieden ze ten opzichte van commerciële MBT-tools? Kunnen kleinere tools ook waarde brengen voor Refleqt, of zijn ze te beperkt in mogelijkheden? Kortom, de open source zijde en de kleinere tools zijn nog niet uitgebreid onderzocht en kunnen mogelijk het test proces van Refleqt efficiënter maken dan wordt aangenomen.

4.3 Persoonlijke reflectie

Dit onderzoek was een eyeopener voor mij. Ik wist dat het concept van MBT niet zo voor de hand liggend was, maar ik besefte niet dat er zoveel manieren zijn voor het genereren van testcases. Natuurlijk wilt een tester niet altijd telkens alle testcases genereren. Soms heeft een tester slechts een klein groepje nodig om zo opnieuw tijd en geld te besparen.

Verder had ik helemaal over het hoofd gezien dat het wel eens wenselijk zou kunnen zijn voor Refleqt indien zo een MBT-tool volledig zou kunnen worden opgenomen binnen de CI cyclus. Het automatisch genereren van testcases is een ding, maar de uitkomst meteen inbrengen in de DevOps cyclus was iets waar ik nog niet over had nagedacht.

Kortom, ik had aangenomen dat MBT ging over het automatisch genereren van testcases op basis van een model, meer niet. Het feit dat er inderdaad ontzettend veel soorten modellen zijn, ontging me. Het model moet namelijk ook kunnen werken met de tool en nadien moet er ook iets bruikbaars uit de tool kunnen komen. Ik had geen idee dat er zoveel kwam kijken bij de verschillende opties van generatie van de testcases.

Conclusie

Het verloop van deze stage-ervaring gebeurde tweeledig. Enerzijds werd gewerkt aan een stageopdracht, hierbij werd kennisgemaakt met nieuwe technologieën op vlak van testing. Anderzijds werd onderzocht wat Model Based-Testing (MBT) inhoudt. Kennismaken met beide technologieën en concepten zorgde voor een unieke stage-ervaring waarbij ontzettend veel werd bijgeleerd.

Voor de stageopdracht werd beroep gedaan op een eerder IT-project. Bij dit project werd een stevige fundering voor de Crap App gelegd. Dit is een trainingsapplicatie waarmee testers zichzelf kunnen trainen op het vinden van bugs. Deze bugs of softwaredefecten kunnen in de app aan- of uitgezet worden. Verder bevat de applicatie draagbaarheid; het kan overal uitgevoerd worden, zoals bij seminaries voor studenten.

Bij de Crap App waren al geautomatiseerde testen opgesteld. Deze testen waren gebouwd zodat ze gewoon zouden werken, aan een toekomstperspectief werd minder aandacht geschonken. Het opruimen van deze tests om ze bruikbaar voor de hele app te maken, vormde de eerste opdracht. Uiteindelijk werd een groot deel code van Cucumber en Selenium geoptimaliseerd en werd de code veel bruikbaarder. Verschillende zaken zijn eenvoudiger terug te vinden en bevinden zich nu op meer logische plaatsen. Een grote geautomatiseerde test opstellen aan de hand van deze verbeterde geautomatiseerde testen, vormde het tweede deel van de stageopdracht. Binnen deze test wordt alle functionaliteit van de loginpagina gecontroleerd.

Het tweede deel van de stage-ervaring houdt een onderzoeksopdracht over Model Based-Testing in. Aangezien dit een volledig nieuw concept was, werd onderzocht wat het precies is en welke tools eventueel waarde zouden kunnen bieden voor Refleqt. MBT is het automatisch genereren van testcases op basis van modellen. Deze modellen stellen het gedrag van een te testen systeem voor. De notatie van deze modellen kan op verschillende manieren gedaan worden, alsook hoe de testcases precies gegenereerd worden. Een globaal overzicht van deze zogenaamde benaderingen of strategieën vertelt dat er mogelijk te veel opties zijn. Als een tester kiest voor een bepaalde MBT-tool, kiest de tester waarschijnlijk ook voor een bepaalde strategie van testgeneratie die de tool aanbiedt. Op die manier kiest de tester voor een tool die het beste aansluit bij de noden van het project of van het bedrijf. Elk project en elk bedrijf zijn tenslotte uniek en vragen andere eigenschappen.

Daarbij komt de vergelijking van MBT-tools kijken. Refleqt gaf bepaalde requirements mee die van belang zijn voor de keuze van een goede MBT-tool voor het bedrijf. Na het vergelijken van Conformiq Creator, MBTsuite en Yest kwam uiteindelijk Conformiq Creator als winnaar uit de bus. Deze tool paste het best bij de gevraagde vereisten van Refleqt.

Bij het onderzoek lag voor de vergelijking de focus op tools die met BPMN, een modeleringstandaard, overweg kunnen. Er zijn nog andere tools op de markt die mogelijk toch nuttig kunnen zijn voor Refleqt, maar die bijvoorbeeld open source werken, of die op veel kleinere schaal werken. Mogelijk is het toch interessant om ook deze tools eens te onderzoeken, omdat deze tools eventueel toch een antwoord kunnen bieden op een niche probleem van Refleqt. Kortom, er is erg veel ter beschikking op de markt op vlak van MBT. Meteen een grote tool met uitermate veel opties inbrengen in de huidige test procedures van Refleqt, is misschien niet het beste idee. Een kleinere tool is misschien sneller ingebracht en brengt mogelijk sneller waarde voor het bedrijf.

Bibliografie

[1] S. &. S. V. Ramasamy, „Researchgate,” 6 juli 2015. [Online]. Available: https://www.researchgate.net/publication/273126286_An_exploration_of_model_based_testing . [Geopend 5 april 2019].

[2] „IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains,” JetBrains, [Online]. Available: https://www.jetbrains.com/idea/. [Geopend 29 april 2019].

[3] „BitBucket | The Git solution for professional teams,” Atlassian, [Online]. Available: https://bitbucket.org/product/. [Geopend 29 april 2019].

[4] „Git,” Software Freedom Conservancy, [Online]. Available: https://git-scm.com/. [Geopend 29 april 2019].

[5] „BDD: Learn about Behavior Driven Development | Agile Alliance,” Agile Alliance, [Online]. Available: https://www.agilealliance.org/glossary/bdd. [Geopend 29 april 2019].

[6] „Vasudha Singh,” Researchgate, [Online]. Available: https://www.researchgate.net/profile/Vasudha_Singh8. [Geopend 9 mei 2019].

[7] „Subburaj Ramasamy,” Researchgate, [Online]. Available: https://www.researchgate.net/profile/Subburaj_Ramasamy2. [Geopend 9 mei 2019].

[8] „Gherkin Reference : Cucumber,” Cucumber, [Online]. Available: https://cucumber.io/docs/gherkin/reference/. [Geopend 8 mei 2019].

[9] „Confluence - Team Collaboration Software | Atlassian,” Atlassian, [Online]. Available: https://www.atlassian.com/software/confluence. [Geopend 8 mei 2019].

[10] „Jira | Issue & Project Tracking Software | Atlassian,” Atlassian, [Online]. Available: https://www.atlassian.com/software/jira. [Geopend 8 mei 2019].

[11] „Overzicht van onderzoeksmethoden en dataverzamelingsmethoden,” Scribbr, [Online]. Available: https://www.scribbr.nl/category/onderzoeksmethoden/. [Geopend 22 maart 2019].

[12] „Types of Test Automation Frameworks | Software Testing Material,” Software Testing Material, [Online]. Available: https://www.softwaretestingmaterial.com/types-test-automation-frameworks/. [Geopend 9 mei 2019].

[13] „What is "Given - When - Then"? | Agile Alliance,” Agile Alliance, [Online]. Available: https://www.agilealliance.org/glossary/gwt. [Geopend 8 mei 2019].

[14] „What is Automation Testing (Ultimate Guide to Start Test Automation),” Software Testing Help, 21 oktober 2018. [Online]. Available: https://www.softwaretestinghelp.com/automation-testing-tutorial-1/. [Geopend 5 april 2019].

[15] Macguy314, PerhelionGerman en Babakus, Artists, *Finite state machine example with comments.svg.* [Art]. Public Domain, Wikimedia, 2007.

[16] „Unified Modeling Language - Wikipedia," Wikipedia, [Online]. Available: https://nl.wikipedia.org/wiki/Unified_Modeling_Language. [Geopend 10 mei 2019].

[17] „BPMN Specification - Business Process Model and Notation," Object Management Group, [Online]. Available: http://www.bpmn.org/. [Geopend 15 mei 2019].

[18] „Jenkins," Jenkins, [Online]. Available: https://jenkins.io/. [Geopend 15 mei 2019].

[19] „Continuous Integration," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Continuous_integration. [Geopend 15 mei 2019].

[20] „Business Proces Model and Notation," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation. [Geopend 15 mei 2019].

[21] „Conformiq Creator | Conformiq Products," Conformiq, [Online]. Available: https://www.conformiq.com/products/conformiq-creator/. [Geopend 16 mei 2019].

[22] „Home | MBTsuite - The Model Based Testing Tool," MBTsuite, [Online]. Available: https://mbtsuite.com/. [Geopend 16 mei 2019].

[23] „YEST - Features | Smartesting," Smartesting, [Online]. Available: https://www.smartesting.com/yest-fonctionnalites?lang=en. [Geopend 16 mei 2019].

[24] „MBTsuite: sepp.med gmbh," sepp.med, [Online]. Available: https://www.seppmed.de/de/portfolio/mbtsuite/. [Geopend 16 mei 2019].

[25] „Unternehmen: sepp.med gmbh," sepp.med, [Online]. Available: https://www.seppmed.de/de/unternehmen/. [Geopend 16 mei 2019].

[26] I. Conformiq, „Conformiq Creator 4.1 Release Overview - YouTube," YouTube, 19 december 2018. [Online]. Available: https://www.youtube.com/watch?v=mXck8GYFXAc&t=147s. [Geopend 17 mei 2019].

[27] I. Conformiq, „Working Effectively with Conformiq Creator Part 1 - YouTube," YouTube, 22 november 2018. [Online]. Available: https://www.youtube.com/watch?v=w2IDSasmvIw&t=2s. [Geopend 17 mei 2019].

[28] s. gmbh, „MBTsuite - Testmanagement - Martin Beißer (Webinar vom 17.12.15) - YouTube," YouTube, 23 december 2016. [Online]. Available: https://www.youtube.com/watch?v=n8uZEAie_ng. [Geopend 17 mei 2019].

[29] „Getting Started | Conformiq," Conformiq, [Online]. Available: https://www.conformiq.com/resources/getting-started/. [Geopend 23 mei 2019].

[30] „Product MBTsuite," Embedded World, [Online]. Available: https://www.embedded-world.de/de/ausstellerprodukte/embwld19/produkt-10135156/mbtsuite. [Geopend 23 mei 2019].

[31] M. Beißer-Dresel, „Home | MBTsuite - The Model Based Testing Tool,” 2019. [Online]. Available: https://mbtsuite.com/. [Geopend 31 mei 2019].

[32] „MoSCoW-methode - Wikipedia,” [Online]. Available: https://nl.wikipedia.org/wiki/MoSCoW-methode. [Geopend 4 juni 2019].

[33] „nice-to-have - Wiktionary,” Wiktionary, [Online]. Available: https://en.wiktionary.org/wiki/nice-to-have. [Geopend 5 juni 2019].

[34] „Versiebeheersysteem - Wikipedia,” Wikipedia, [Online]. Available: https://nl.wikipedia.org/wiki/Versiebeheersysteem. [Geopend 5 juni 2019].

[35] „Systems development life cycle - Wikipedia,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Systems_development_life_cycle. [Geopend 5 juni 2019].

[36] S. Dalal, A. Jain, N. Karunanithi, J. Leaton, C. Lott, G. Patton en B. Horowitz, „Model-based testing in practice - IEEE Conference Publication,” 22 mei 1999. [Online]. Available: https://ieeexplore.ieee.org/document/841019. [Geopend 8 juni 2019].

[37] M. Utting, A. Pretschner en B. Legeard, „A taxonomy of model-based testing approaches - Utting - 2012 - Software Testing, Verification and Reliability - Wiley Online Library,” 12 april 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.456 . [Geopend 8 juni 2019].

[38] Conformiq, „What-You-Need-to-Consider-When-Selecting-an-MBT-Tool-Whitepaper,” 2015. [Online]. Available: https://www.conformiq.com/wp-content/uploads/2014/12/What-You-Need-to-Consider-When-Selecting-an-MBT-Tool-Whitepaper.pdf. [Geopend 9 juni 2019].

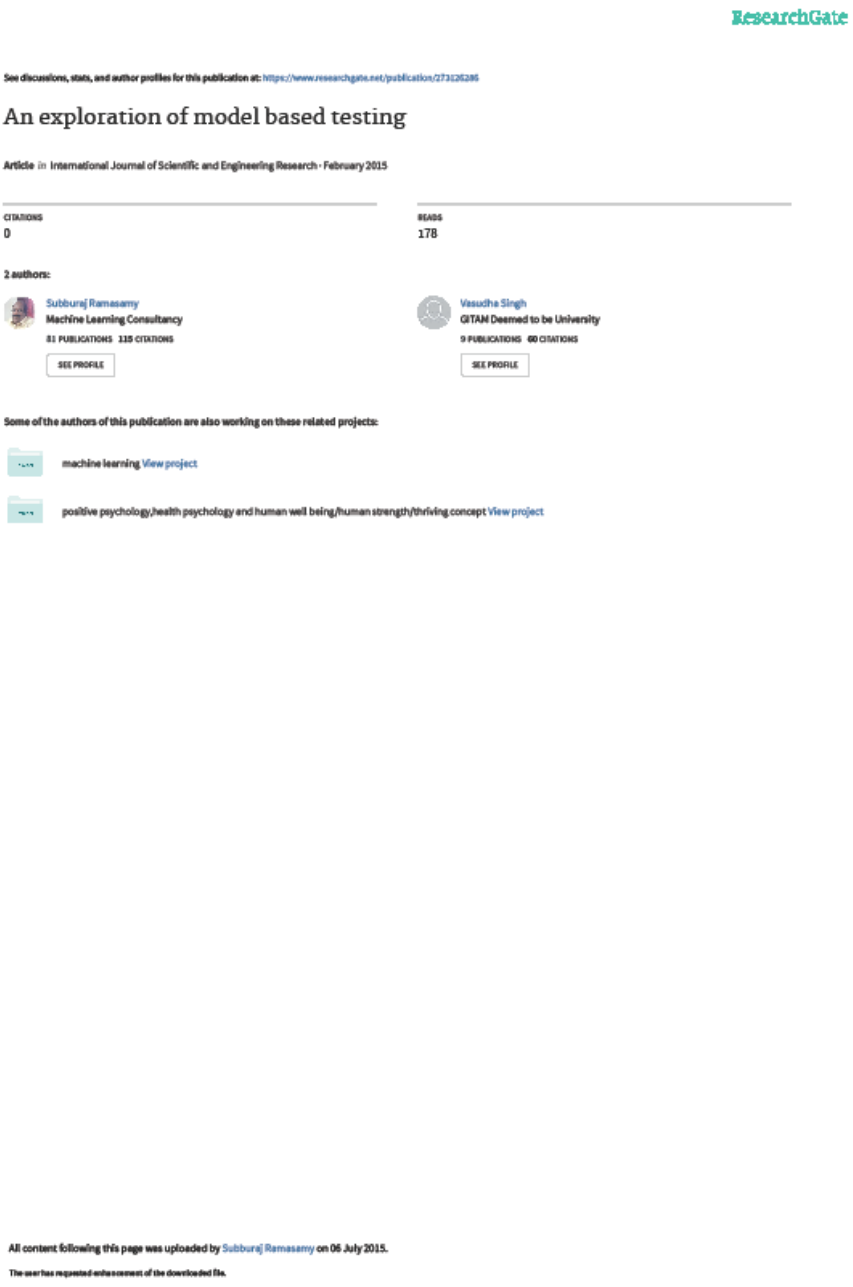
[39] „indeterminisme - 2 definities - Encyclo,” Encyclo, [Online]. Available: https://www.encyclo.nl/begrip/Indeterminisme. [Geopend 9 juni 2019].

Bijlage

Bijlage A	Researchwerk *“An Exploration of Model Based-Testing”*
Bijlage B	Code van de grote test: LoginPageFunctionality.feature
Bijlage C	*Conference paper "Model-based testing in practice”*
Bijlage D	Onderzoekswerk *“A taxonomy of model-based testing approaches”*

A. Omschrijving Bijlage A

Link naar PDF-bestand:
https://www.researchgate.net/publication/273126286_An_exploration_of_model_based_testing



ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/273126286

An exploration of model based testing

Article in International Journal of Scientific and Engineering Research · February 2015

CITATIONS
0

READS
178

2 authors:

Subburaj Ramasamy
Machine Learning Consultancy
81 PUBLICATIONS 115 CITATIONS
SEE PROFILE

Vasudha Singh
GITAM Deemed to be University
9 PUBLICATIONS 60 CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

machine learning View project

positive psychology,health psychology and human well being,human strength/thriving concept View project

All content following this page was uploaded by Subburaj Ramasamy on 06 July 2015.

The user has requested enhancement of the downloaded file.

B. Omschrijving Bijlage B

LoginPageFunctionality.feature

Code van de grote test:

Hieronder staat de code zelf. Houd er rekening mee dat de insprong van de code op verschillende plaatsen niet klopt en dus een erg verwarrend beeld weergeeft. Het beste wordt de code bekeken in een programma zoals IntelliJ IDEA of in een tekstverwerkingsprogramma zoals Notepad++.

```
@Sanity
Feature: Functionality of the login page

  Background: Shared steps of registration, login, forgot username and
forgot password functionality
    Given a user is on the login page

  Scenario Outline: Registration attempts
    When the user clicks the REGISTER button on the login page
    And the user enters the email address <email> on the registration page
    And the user enters the username <username> on the registration page
    And the user selects the gender <gender> on the registration page
    And the user enters the password <password> on the registration page
    And the user enters the confirmation password <confirmationPassword> on
the registration page
    And the user clicks the registration page register button
    Then the user gets the registration message <expectedMessage>

    Examples:
    | email        | username       | gender | password        |
confirmationPassword | expectedMessage |
    | validEmail   | validUsername | male   | validPassword   |
matchingPassword     | none             |
    | validEmail   | validUsername | female | validPassword   |
matchingPassword     | email is required |
    | invalidEmail | validUsername | male   | validPassword   |
matchingPassword     | Insert a valid email |
    | takenEmail   | validUsername | female | validPassword   |
matchingPassword     | Email is taken  |
    | validEmail   | validUsername | male   | empty           | empty
| Password is required |
    | validEmail   | validUsername | female | invalidPassword |
matchingPassword     | Password is not a valid password your password must
at least: |
    | validEmail   | validUsername | male   | validPassword   |
nonMatchingPassword  | Passwords do not match |
    | validEmail   | takenUsername | female | validPassword   |
matchingPassword     | Username is taken |
    | validEmail   | empty         | male   | validPassword   |
matchingPassword     | Username is required |

  Scenario: Cancel registration
    When the user clicks the cancel button on the registration page
    Then the browser navigates to the login page

#  Voor demo: bij scenario 2: 'empty' is correct, 'validEmail' geeft een
gefaalde test terug

#  Testen van de login funtionaliteit
```

B1

B2

```
  Scenario Outline: Login attempts
    When the user enters the username <username> on the login page
    And the user enters the password <password> on the login page
    And the user clicks the login button on the login page
    Then the user gets the login message <expectedMessage>

    Examples:
      | username        | password        | expectedMessage |
      | validUsername   | validPassword   | none            |
      | validUsername   | invalidPassword | password is not a valid
password |
      | validUsername   | wrongPassword   | password is wrong |
      | validUsername   | empty           | password is required |
      | invalidUsername | validPassword   | user does not exist  |
      | invalidUsername | invalidPassword | password is not a valid
password |
      | invalidUsername | wrongPassword   | user does not exist  |
      | invalidUsername | empty           | password is required |
      | empty           | validPassword   | username is required |
      | empty           | invalidPassword | username is required and
password is not a valid password |
      | empty           | wrongPassword   | username is required |
      | empty           | empty           | username is required and
password is required |

# wrongPassword: password is een correct opgesteld ww, maar hoort niet bij
de gebruiker
# invalidPassword: ww is niet correct opgesteld (niet lang genoeg, bevat
geen speciaal teken, etc)

#  'I forgot my password' functionaliteit

  Scenario Outline: Forgotten password attempts
    When the user clicks the 'I Forgot my password' button on the login
page
    And the user enters the email <email> on the forgot password page
    And the user clicks the send button on the forgot password page
    Then the user gets the forgot password message <expectedMessage>

    Examples:
      | email        | expectedMessage   |
      | validEmail   | none              |
      | empty        | email is required |
      | invalidEmail | email not found   |

#  'I forgot my username' functionaliteit

  Scenario Outline: Forgotten username attempts
    When the user clicks the 'I Forgot my username' button on the login
page
    And the user enters the email <email> on the forgot username page
    And the user clicks the send button on the forgot username page
    Then the user gets the forgot username message <expectedMessage>

    Examples:
      | email        | expectedMessage   |
      | validEmail   | none              |
      | empty        | email is required |
      | invalidEmail | email not found   |
```

B3

B4

B5

C. Omschrijving Bijlage C

Dit is de bron van deze bijlage: [36]. Om effectief deze paper te kunnen raadplegen, wordt de volgende werkwijze gevolgd. Eerst wordt in Google gezocht naar 'Model based testing'. Daarna wordt bovenaan op 'Wetenschappelijke artikelen voor Model based testing' geklikt. Vervolgens is ergens tussen de zoekresultaten een link te zien genaamd 'Model -based testing in practice'. Rechts van deze link staat een aparte link voor de pdf-versie van de paper. Zo kan dus de paper geraadpleegd worden.

Hierbij een link naar de zoekresultaten, het derde resultaat is de correcte bron:
https://scholar.google.be/scholar?q=model-based+testing&hl=nl&as_sdt=0&as_vis=1&oi=scholart

To appear in Proceedings of ICSE'99, May 1999 (ACM Press).

Model-Based Testing in Practice

S. R. Dalal, A. Jain, N. Karunanithi,
J. M. Leaton, C. M. Lott, G. C. Patton
Bellcore
445 South Street
Morristown NJ 07960, USA
{sid, jain, karun, jleaton, lott, gcp}@bellcore.com

B. M. Horowitz
Bellcore
6 Corporate Place
Piscataway NJ 08854, USA
bruceh@cc.bellcore.com

ABSTRACT
Model-based testing is a new and evolving technique for generating a suite of test cases from requirements. Testers using this approach concentrate on a data model and generation infrastructure instead of hand-crafting individual tests. Several relatively small studies have demonstrated how combinatorial test generation techniques allow testers to achieve broad coverage of the input domain with a small number of tests. We have conducted several relatively large projects in which we applied these techniques to systems with millions of lines of code. Given the complexity of testing, the model-based testing approach was used in conjunction with test automation harnesses. Since no large empirical study has been conducted to measure efficacy of this new approach, we report on our experience with developing tools and methods in support of model-based testing. The four case studies presented here offer details and results of applying combinatorial test-generation techniques on a large scale to diverse applications. Based on the four projects, we offer our insights into what works in practice and our thoughts about obstacles to transferring this technology into testing organizations.

Keywords
Model-based testing, automatic test generation, AETG software system.

1 INTRODUCTION
Product testers, like developers, are placed under severe pressure by the short release cycles expected in today's software markets. In the telecommunications domain, customers contract for large, custom-built systems and demand high reliability of their software. Due to increased competition in telecom markets, the customers are also demanding cost reductions in their maintenance contracts. All of these issues have encouraged product test organizations to search for techniques that improve upon the traditional approach of hand-crafting individual test cases.

Test automation techniques offer much hope for testers. The simplest application is running tests automatically. This allows suites of hand-crafted tests to serve as regression tests. However, automated execution of tests does not address the problems of costly test development and uncertain coverage of the input domain.

We have been researching, developing, and applying the idea of automatic test generation, which we call model-based testing. This approach involves developing and using a data model to generate tests. The model is essentially a specification of the inputs to the software, and can be developed early in the cycle from requirements information. Test selection criteria are expressed in algorithms, and can be tuned in response to experience. In the ideal case, a regression test suite can be generated that is a turnkey solution to testing the piece of software: the suite includes inputs, expected outputs, and necessary infrastructure to run the tests automatically.

While the model-based test approach is not a panacea, it offers considerable promise in reducing the cost of test generation, increasing the effectiveness of the tests, and shortening the testing cycle. Test generation can be especially effective for systems that are changed frequently, because testers can update the data model and then rapidly regenerate a test suite, avoiding tedious and error-prone editing of a suite of hand-crafted tests.

At present, many commercially available tools expect the tester to be 1/3 developer, 1/3 system engineer, and 1/3 tester. Unfortunately, such savvy testers are few or the budget to hire such testers is simply not there. It is a mistake to develop technology that does not adequately address the competence of a majority of its users. Our efforts have focused on developing methods and techniques to support model-based testing that will be adopted readily by testers, and this goal influenced our work in many ways.

We discuss our approach to model-based testing, including some details about modeling notations and test-selection algorithms in Section 2. Section 3 surveys related work. Four large-scale applications of model-based testing are presented in Section 4. Finally, we offer some lessons learned about what works and does not work in practice in Section 5.

D. Omschrijving bijlage D

Deze bron is via bron [37] officieel terug te vinden. Om de pdf rechtstreeks te kunnen raadplegen, dient deze link gebruikt te worden:

http://eprints.qut.edu.au/57853/1/master_pdflatex.pdf