# Testautomatisering mobiele applicaties

Ontwerpen en implementeren van een testframework die het automatisch acceptatietesten van mobiele applicaties mogelijk maakt binnen een ‘continuous integration’ omgeving.


Rick I. S. de Hoop
6 oktober 2014
Afstudeerverslag HBO Informatica

# Gegevens

## Titel

Testautomatisering mobiele applicaties

Ontwerpen en implementeren van een testframework die het automatisch testen van mobiele applicaties mogelijk maakt binnen een 'continuous integration' omgeving.


## Auteur

Rick I. S. de Hoop
Studentennummer 10102094
E-mailadres rickdehoop@gmail.com
Telefoonnummer +31 (0) 6 49 61 68 31


## Begeleiders

Remco Ruijsenaars, begeleidend examinator, de Haagse Hogeschool
Emeri Koenen, tweede examinator, de Haagse Hogeschool
Evan Kluin, bedrijfsmentor, IceMobile

## Opdrachtgevers

Desmond Delissen, opdrachtgever, IceMobile
Bert Rijsdijk, opdrachtgever, IceMobile

## Studie

Opleidingsinstituut de Haagse Hogeschool
Locatie Zoetermeer
Opleiding Informatica

## Opdrachtgever

IceMobile
Website www.icemobile.com
Telefoonnummer +31 (0) 20 368 06 45

Amsterdam, 6 oktober 2014

# Voorwoord

Mijn afstudeerstage bij IceMobile was enorm leerzaam. Het zijn mijn laatste punten voor mijn HBO bachelor en ik kan niet wachten om aan het werk te gaan in het bedrijfsleven. Ik heb een geweldige tijd bij IceMobile gehad, zowel qua leren als in alles daar omheen.

Desmond vroeg mij of ik een testframework wilde ontwikkelen voor mijn afstudeerstage. Ik had geen idee wat dat op dat moment inhield maar het leek mij zeer interessant. Ik dank Desmond dan ook voor het bedenken van de opdracht.

Daarnaast wil ik iedereen bij IceMobile bedanken voor deze geweldige afstudeerplek. In het bijzonder dank ik Evan, vriend en stagebegeleider, voor alle hulp tijdens mijn werkzaamheden en goede tijden buiten werk om.

Dit zijn de laatste loodjes van mijn opleiding. Al mijn voorgaande projectgenootjes van de HHS, vrienden en familie dank ik voor jullie steun, zonder jullie was ik hier nooit geraakt.

Tot slot wil ik mijn ouders bedanken die het mogelijk hebben gemaakt om te studeren. Heel erg bedankt!

Schroom niet om contact te zoeken met mij voor eventuele vragen of opmerkingen. Veel plezier tijdens het lezen van dit verslag!

Rick de Hoop, Woerden
Zondag 5 oktober 2014

## Samenvatting

De opdracht bestaat uit het ontwikkelen van een platform waarmee de mobiele applicaties, die worden ontwikkeld door IceMobile, automatisch kunnen worden getest aan de hand van een door een tester opgestelde test-flow. In de huidige situatie worden de test-flows nog handmatig uitgevoerd door de testers. Daarnaast moet dit platform in een continuous integration (CI) omgeving geplaatst kunnen worden.

Vanwege de beperkte kennis over dit onderwerp bij zowel de afstudeerder als bij de opdrachtgevers is er onderzoek gedaan naar de verschillende oplossingen die mogelijk zijn: alles zelf ontwikkelen, gebruik maken van bestaande frameworks of een combinatie hiervan. Er is gekozen voor een combinatie.

Er is gewerkt met de Agile softwareontwikkelmethode scrum. Er is allereerst een testframework ontwikkelt waarmee de mobiele applicaties automatisch kunnen worden getest. Tijdens deze fase kwam aan het licht dat veel gebruikte frameworks nog erg jong zijn en er daarom nog een aantal vervelende bugs bezitten. Dit zijn problemen waar veel aan is gewerkt in deze fase. Later is er tijdens het project nog een belangrijke switch gemaakt naar een ander framework wat er voor zorgde dat er een aantal dingen intern anders werkten. De input is nog steeds een test en de output is nog steeds een testrapport. Dus wat dat betreft is er niet veel veranderd.

Met een testframework als basis kunnen er testen worden geschreven en kunnen deze worden geautomatiseerd. Echter worden ze nog steeds met de hand gestart. Het testframework is in een CI-omgeving geplaatst. Dat wil zeggen dat nadat een ontwikkelaar zijn werk (code) naar de servers heeft gestuurd het hele project automatisch wordt gecompileerd en getest. Daarna worden de resultaten in een overzicht weergegeven met een screenshot van het moment van falen (áls de test is gefaald) en een knop om het gefaalde scenario eventueel te melden bij de projectmanagers en ontwikkelaars. Op deze manier heeft de tester ten alle tijden de testresultaten bij de hand en kan de tester snel en doelgericht een nieuwe melding van een bug maken.

Er is een testframework opgeleverd die zowel lokaal als in een CI-omgeving kan draaien. Het testframework biedt standaard ondersteuning voor iOS, Android en Firefox OS maar kan in principe met elk testframework worden gecombineerd. Daarnaast is de hele CI-server opgeleverd inclusief handleiding om de server vanaf de grond af aan opnieuw op te bouwen.

De opgeleverde producten zijn op het moment van publiceren in pilotfase bij IceMobile en wordt naar waarschijnlijkheid doorontwikkeld.

# Inhoudsopgave

# 1 Inleiding

Als afsluitend onderdeel van mijn opleiding Informatica aan de Haagse Hogeschool moet ik afstuderen. Dit onderdeel bestaat uit een afstudeerstage bij IceMobile met een bijpassende afstudeeropdracht. IceMobile bestaat 12 jaar. IceMobile is vanaf de oprichting gericht op mobiele technologie.

Mijn opdracht bestaat uit het ontwikkelen van een platform waarmee de mobiele applicaties, die worden ontwikkeld door IceMobile, automatisch kunnen worden getest aan de hand van een door een tester opgestelde test-flow. In de huidige situatie worden de test-flows nog handmatig uitgevoerd door de testers.

Het doel van dit project is een platform ontwikkelen waarmee de front end van IceMobile's mobiele applicaties automatisch kan worden getest zonder handmatig stappen te moeten doorlopen. Deze stappen moeten één keer worden beschreven in begrijpelijke taal voor het gehele bedrijf. Daarnaast moet de implementatie van de stappen niet ingewikkeld zijn zodat er snel door verschillende mensen een test kan worden geïmplementeerd. Als laatste moet dit platform worden geplaatst binnen een continuous integration omgeving met automatische testaansturing en bug-meldingen naar de issue tracking systemen.

Dit verslag is bedoeld voor de Haagse Hogeschool en iedereen die interesse heeft in de doorlopen stappen en gemaakte keuzes om de eerdergenoemde doelen te behalen.

Allereerst wordt er een korte beschrijving gegeven van IceMobile waarna vervolgens de projectaanpak wordt beschreven. Vervolgens is er een hoofdstuk aanwezig die het systeem abstract beschrijft ter verduidelijking voor de lezer. In de daar op volgende hoofdstukken worden, per fase, de volgende punten beschreven: gebruikte software, werkwijze, keuzeonderbouwing, problemen, oplossingen, keuzeverantwoording, samenhang met de productbeschrijving en tijdinvestering. In de laatste hoofdstukken kunt u een evaluatie vinden van het doorlopen proces en de opgeleverde product(en).

# 2 IceMobile

*IceMobile is nog niet een heel oud bedrijf maar is in het vakgebied waarin zij actief zijn een pionier te noemen. De geschiedenis van IceMobile is beschreven in paragraaf 2.1. De organisatiestructuur is beschreven in paragraaf 2.2. IceMobile heeft een aantal doelstellingen waar naar wordt gestreefd. Daarnaast heeft dit bedrijf ook een aantal motto's die de manier van werken vormgeven (paragraaf 2.3).*

## 2.1 Geschiedenis

IceMobile is opgericht in 2002 met de visie dat de mobiele apparaten het meest belangrijke medium worden in het leven van de mens. In die tijd waren er nog geen smartphones en was het mobiele internet nog niet eens gelanceerd, er waren wel protocollen maar die werden simpelweg nog niet gebruikt. Ter vergelijking, de eerste iPhone is pas 5 jaar na oprichting van IceMobile gelanceerd en vele zien dit als het echte begin van de smartphone zoals die nu bekend is. De founders van IceMobile zien de smartphone nog steeds niet zoals zij het willen zien, sterker nog, IceMobile denkt dat het nu pas echt interessant gaat worden.

In de eerste jaren was IceMobile vooral technisch geaard. De opdrachten die werden binnengehaald waren niet concreet en vaak meer een Proof Of Concept dan een werkelijk product. Bijvoorbeeld: "Wij als groot sportmerk willen graag iets met mobiel. Kunnen jullie iets maken voor ons?". Deze opdracht heeft zich vertaald in een reclamefilmpje dat was te downloaden naar de mobiele telefoon over het mobiele netwerk. Het sportmerk heeft dit laten zien tijdens een grote presentatie om aan te tonen dat zij, het sportmerk, niet stil staan en meegaan met hun tijd. Niet alleen met hun producten maar ook hun marketingbeleid. Het filmpje zelf is echter nog geen 20 keer gedownload. Want wie wil er betalen voor een reclamefilmpje en het vervolgens bekijken op een scherm met een zeer kleine pixeldichtheid?

Het viel IceMobile tegen hoelang het duurde voordat mobiel echt doorbrak. Ze waren er erg vroeg bij. Naar mate de tijd verstreek groeide het aantal mobiele telefoons bij de consumenten. De opdrachten werden ook specifieker. "Kunnen jullie iets maken waarmee MMS'jes via de post worden verstuurd als kaart?". Dit bleek technisch haalbaar maar niet bruikbaar want de camera's van de mobiele telefoons van toen waren zeer onscherp. Uiteindelijk werden de kaarten vervangen door kleine stickers.

Vanaf dat moment kwam het mobiele internet steeds meer in beeld met bijvoorbeeld WAP, iMode van KPN en al snel de Symbian telefoons van Nokia. Maar alles raakte pas echt in een stroomverstelling toen de iPhone

uitkwam. Vanaf dat moment is IceMobile ook direct aan de slag gegaan met app-ontwikkeling. Appie, van Albert Heijn, was de eerste echte grote opdracht van IceMobile. Er volgden al snel vele opdrachten van grote Nederlandse bedrijven omdat zij de enige partij in het hele land waren die deze service aanboden.

De app-ontwikkeling maakte van IceMobile opeens een heel ander bedrijf. Niet langer was alleen de techniek nog belangrijk maar ook de vormgeving en UX werd opeens heel belangrijk. IceMobile is al snel gefuseerd met een bedrijf, Muse, dat gespecialiseerd was in interaction design en vormgeving. De combinatie van deze 2 bedrijven maakt IceMobile, tot op heden, ontzettend sterk. Ook hier waren zij de enige in het hele land met de combinatie van techniek en innovatie/design.

IceMobile wilde doorgroeien maar zag eigenlijk alleen nog maar mogelijkheden in het buitenland aangezien bijna alle grote bedrijven binnen Nederland al klant waren van IceMobile. Echter zagen de eigenaren en werknemers het niet zitten om te verhuizen naar het buitenland en bedachten ze zich tegelijkertijd dat het moeilijk concurreren zou gaan worden met bedrijven die dichterbij de klant zijn gevestigd.

Het werd dus tijd om zich te gaan specialiseren. En zo werd BrandLoyalty, een Nederlands bedrijf, gevonden die wereldwijd loyaliteitsprogramma's verkoopt. BrandLoyalty is nog van de oude stempel, met de conventionele spaarkaarten en losse zegeltjes, en zou graag een mobiele versie van hun spaarboekjes/zegelkaarten willen zien. Deze twee bedrijven zijn samen de samenwerking aangegaan en in oktober 2013 was de digitale versie klaar om in de verkoop te gaan. Dit is nu dan ook de main-focus van IceMobile. Alleen ABN Amro is nog behouden als klant.

Het bedrijf staat niet stil en is nu bezig met 'IT-Free'. IceMobile wil zoveel mogelijk klanten de digitale zegeltjes aanbieden, echter kost dit veel tijd voor de IT-afdelingen van de supermarktketens. De supermarktketens moeten namelijk ook hun systemen klaarmaken voor het digitale sparen. Met IT-Free wil IceMobile de gehele IT-afdeling ontlasten door helemaal geen connecties te hebben met deze afdeling. Op die manier is de drempel voor een supermarktketen kleiner en ook meteen een ingang om het grote pakket aan te bieden (met integratie van de systemen van de klanten).

IceMobile is sinds de oprichting gegroeid van 2 mensen naar ongeveer 120 mensen. Sinds februari 2013 is IceMobile echter onwijs snel gegroeid en doet dat nog steeds.

## 2.2 Organisatie

Wat meteen opvalt als je het gebouw betreedt is het oog voor presentatie. Alles ziet er verzorgd uit. Dat geldt ook voor de werknemers, zij lopen niet in pak maar wel verzorgd. Zolang de werknemers er verzorgd uit zien zijn er eigenlijk geen kledingvoorschriften. Slippers zijn dan ook gewoon toegestaan tijdens de warme zomerdagen zolang men er maar goed bijloopt.

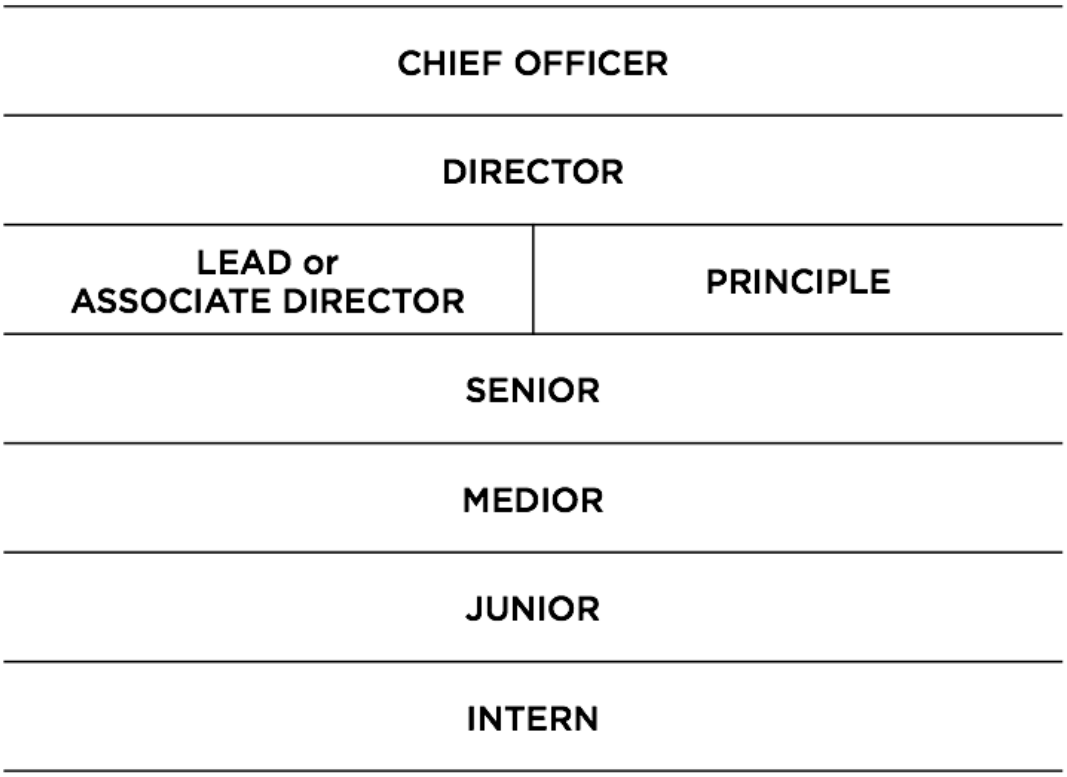
Als men het bedrijfspand verder doorloopt zal men opvallen dat er geen losse cubes zijn waar mensen werken maar iedereen in grotere ruimtes zit. Er zijn een aantal vergaderruimtes die kunnen worden gereserveerd voor meetings maar deze zijn ook vaak voorzien van glas. De indeling van het bedrijfspand is per project en niet per discipline (op het management na). Op die manier heb je altijd de juiste mensen om je heen en kan er snel worden gecommuniceerd met elkaar.

Binnen IceMobile worden er 21 verschillende talen gesproken. Er heerst, onder andere, vanwege de vele verschillende achtergronden een open sfeer. En er zijn naast werk veel verschillende groepen ontstaan waar werknemers samen als vrienden deelnemen aan activiteiten. Dit loopt uiteen van squashen of wandklimmen tot een IceMobile bandje of een eigen mini bierbrouwerij.

Elke vrijdagmiddag/avond is er een borrel met hapjes en drankjes. Soms is er een thema, soms een company-update en soms hebben een aantal werknemers de borrel voorbereid. Daarnaast is er elk jaar een ski-trip voor alle IceMobile medewerkers en wordt er naast skiën ook goed gefeest.

IceMobile heeft een platte organisatiestructuur. In ‘Afbeelding 2-1 Organisatiestructuur IceMobile’ staan de verschillende functiegroepen van IceMobile beschreven. Per discipline geldt deze structuur, iOS-development, NodeJS-development, grafische vormgeving et cetera. Elke werknemer heeft één van de volgende functies: stagiair, junior, medior of senior (behalve het management team). De functies lead, associate en principle worden uitgevoerd naast de eerdergenoemde functies. Deze functies worden eigenlijk altijd door seniors uitgevoerd en zijn expertfuncties. De functie director is de hoogst haalbare functie als gewone medewerker en houdt in dat de medewerker beslissingen kan nemen over problemen van hoog abstract niveau. De chief officer wordt ingevuld door een persoon van het management team en is verantwoordelijk voor de gehele disciplinegroep.

Binnen IceMobile zijn er nu negen van deze disciplinegroepen.




Afbeelding 2-1 Organisatiestructuur IceMobile

## 2.3 Doelstellingen

Binnen IceMobile zijn er een aantal doelstellingen net als ieder ander bedrijf. Zo heeft het bedrijf aan het begin van 2013 een nieuwe weg ingeslagen: specialisatie/productising. De middellange termijndoelstelling is uitbreiding op wereldschaal en dit is intussen in volle gang door de samenwerking met BrandLoyalty en na de recente overname door Alliance Data Systems. Door de samenwerking met deze partijen zijn er opeens veel potentiële klanten bijgekomen. Want beide partijen hebben wereldwijd al vele klanten voor hun loyaliteitsprogramma's. Op de korte termijn is het de bedoeling om veel klanten het nieuwe product te verkopen en deze producten op maat te maken voor de klant. Daarnaast wil IceMobile blijven innoveren. Daarom wordt er continu gewerkt aan nieuwe modules voor het product. Deze modules kunnen afzonderlijk van elkaar aan of uit worden gezet. Uiteindelijk moet dit product het paradepaardje worden van IceMobile.

Om deze doelstellingen te halen wordt er gewerkt met een aantal motto's op de voorgrond.

*Focus on people and all else will follow*, mensen zijn de basis van IceMobile en gemotiveerde en inspirerende mensen leiden tot goede producten. IceMobile vindt het belangrijk om eerst goede producten te maken alvorens er aan winst wordt gedacht.

*Change is the only constant*, met de snelle veranderingen van tegenwoordig is snel reageren relevant. Er wordt veel geëxperimenteerd en er is geen angst om de status quo uit de weg te gaan. Deze vele veranderingen zijn ook erg merkbaar in de cultuur binnen IceMobile.

*Simplicity is the ultimate sophistication*, Om aan de top en belangrijk te blijven is het belangrijk om dingen te versimpelen. Kleine dingen maken het verschil en vele kleine dingen kunnen een ervaringen veranderen op een grote manier. Ook de producten zelf moeten makkelijk in gebruik zijn.

*Magic happens at the intersection*, design en technologie worden gecombineerd in multidisciplinaire teams. De sleutel van de juiste gebruikerservaring ligt op de kruising van design en technologie.

*Do epic shit*, maak coole dingen die pakkend zijn. Zet doelen die onmogelijk lijken en verbaas jezelf. Creëer een betere wereld en maak plezier.

# 3 Opdracht

*Alle projecten beginnen met een opdracht. Om een opdracht goed te beschrijven is het van belang om te weten wat de aanleiding van de opdracht is, welke doelstellingen en resultaten er moeten worden behaald.*

## 3.1 Aanleiding

Om de verkooptargets van de komende jaren te halen moet de implementatietijd per verkocht product worden verkort. Binnen heel IceMobile worden processen geoptimaliseerd.

De opdrachtgevers, Desmond Delissen en Bert Rijsdijk, vertegenwoordigers van IceMobile, zijn verantwoordelijk voor het testen van de door IceMobile ontwikkelde software. Binnen het testproces gebeurt nu nog veel handmatig. De opdrachtgevers willen de benodigde tijd om te testen verkorten door middel van testautomatisering in een continuous integration environment.

De opdrachtgevers geven aan te experimenteren met een aantal testframeworks voor unit-testen en integratietesten. Zij kunnen met behulp van deze frameworks een groep van testen in één keer uitvoeren en in een overzicht aflezen welke testen er zijn gefaald en waarom.

De opdrachtgever is daarom zoekende naar een manier om alle verschillende testen op een zo uniforme en efficiënte manier uit te voeren.

## 3.2 Achtergrond

IceMobile is inmiddels anderhalf jaar bezig met een vernieuwd platform voor digitale spaarzegeltjes (Stamps) en één voor supermarkten in het algemeen (Shopper). Deze platformen, die uiteindelijk beschikbaar worden gesteld door middel van een mobiele applicatie aan de consument, zijn inmiddels dusdanig groot en complex dat testen steeds een belangrijkere rol gaat krijgen. Het aantal bugs neemt snel af maar de te testen onderdelen nemen bij elke versie toe. Regressietesten is daarom het toverwoord.

Regressietesten neemt, als dit handmatig gebeurd, veel tijd in beslag en lijkt vanwege de repetitieve handelingen zeer geschikt voor automatisering.

Bij aanvang van dit project wordt er binnen IceMobile geëxperimenteerd met testautomatisering. Er wordt gebruik gemaakt van Yadda en Mocha als testaansturing. Met deze twee testframeworks kunnen testen worden geschreven en vervolgens worden uitgevoerd met een simpel commando. Het testteam is tevreden met deze twee frameworks en zou dit ook willen

gebruiken voor de front-end van de mobiele applicaties. Daarnaast zou het testteam dit in een continuous integration omgeving willen plaatsen zodat het testsysteem het testteam waarschuwt wanneer er een deel van de applicatie niet meer naar behoren werkt.

## 3.3 Doelstellingen

Het doel van dit project is het automatiseren van het regressietesten voor de front end van de mobiele applicaties van IceMobile. IceMobile heeft op het moment van schrijven Android en iOS applicaties die getest moeten worden. Echter, met het oog op de toekomst wil IceMobile een testautomatiseringsframework die geschikt moet zijn voor nieuwe platformen.

Half september moet er een product worden opgeleverd die automatisch een test-flow voor mobiele applicaties kan uitvoeren die is opgesteld door een tester. De tester, of programmeur, moet deze test-flows vervolgens kunnen implementeren mits deze in ieder geval een minimale kennis heeft van programmeren.

De testen worden gemaakt op een desktop en worden vanzelfsprekend uitgevoerd op een mobiel apparaat en/of een simulator/emulator.

## 3.4 Resultaten

De te behalen resultaten kunnen worden opgedeeld in twee categorieën. Namelijk gewenst resultaat en ongewenst resultaat. Dit is vastgesteld aan het begin van het project door de opdrachtgevers.

### 3.4.1 Gewenst resultaat

Het gewenste resultaat is een testframework ...

- waarmee testen in een taal begrijpelijk voor het management board, of een klant kunnen, worden opgesteld. Denk aan een testscenario die wordt beschreven met zinnen als, ‘gegeven, er is ingelogd’, ‘waarna er op het hamburger-menu-icoontje wordt geklikt’, ‘dan zie je het menu’
- waar de testbeschrijvingen met een minimale programmeerkennis kunnen worden geïmplementeerd.
- Waarbij de te doorlopen teststappen aan de hand van de elementen zelf werken en niet de locatie op het scherm van de elementen. Bij een verplaatsing van de knop mag de test dus niet falen.
- waarbij een kleine verandering in de test-flow niet er niet voor zorgt dat het gehele testscenario opnieuw moet worden geïmplementeerd maar alleen die kleine verandering.

- die in de huidige continuous integration omgeving van IceMobile kan worden geplaatst.

### 3.4.2 Ongewenst resultaat

Ongewenst resultaat kan op verschillende manieren tot stand komen. De volgende scenario's zijn voor dit project het meest van belang:

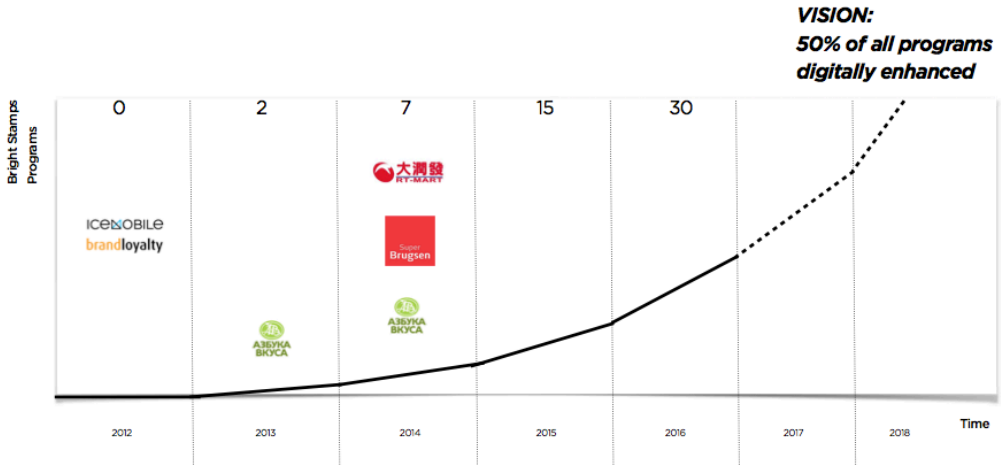
- De opdrachtgever is niet tevreden met een gekozen oplossing. Vanwege de agile ontwikkelmethode (deelparagraaf 5.2.1) zal dit slechts voor vertraging zorgen omdat er snel geschakeld kan worden vanwege tussentijdse demo's.
- De opdrachtgever beoordeelt het werk van onvoldoende kwaliteit. Doordat er met sprints wordt gewerkt hoeft dit niet een groot probleem te zijn. Als dit een systematisch probleem is moet het project misschien verder worden afgebakend of extra hulp komen.
- Er zijn niet voldoende resources beschikbaar. Bij vragen en problemen moet de afstudeerder de mogelijkheid hebben om deze voor te kunnen leggen bij een expert. Als dit niet kan zal het project uitlopen of minder features bevatten vanwege tijdgebrek.
- De beschikbare frameworks benodigd voor dit project zijn nog niet voldoende ontwikkeld. Als dit gebeurd zal er overgestapt moeten worden op een ander framework.
- Het project wordt niet volledig afgerond vanwege onverwachte tegenslagen. Er dient na elke sprint daarom een werkend product te worden opgeleverd.

# 4 Oorspronkelijke situatie

*Bij IceMobile wordt gebruik gemaakt van een aantal vormen van testen. Zo wordt er onder andere aan unit-testing gedaan bij de mobiele apps zelf. Dit gebeurt nu niet voor alle methoden van de applicaties. Ook worden er een groot aantal unit-testen geschreven voor de API-calls. Als de API goed in elkaar zit en de in- en output goed worden afgevangen scheelt dat al een groot aantal bugs. De API is slechts een deel van de applicatie en daarom is het belangrijk dat de mobiele applicaties uitgebreider getest gaan worden.*

## 4.1 Stamps

Nu IceMobile zich steeds meer gaat richten op Stamps moet de implementatietijd van Stamps per klant terug worden gebracht. In 2013 liepen er 2 programma's van Stamps. Dit jaar, 2014, worden er na verwachting 7 programma's gestart met Stamps. De komende 4 jaar moeten deze cijfers elk jaar verdubbelen. Dit houdt in dat er in 2016 per programma een kleine 2 weken is gereserveerd voor Stamps, waar dat er nu nog 8 zijn ('Afbeelding 4-1 Productimplementaties over de tijd').




Afbeelding 4-1 Productimplementaties over de tijd

Parallel aan de klanten implementaties van Stamps moet het product ook worden blijven ontwikkeld. Niet alleen Stamps moet worden voorzien van nieuwe features, IceMobile streeft naar innovatie op meerdere vlakken in bijvoorbeeld de vorm van een nieuw product. Hoe minder tijd IceMobile bezig is met implementaties van Stamps voor klanten des te meer tijd er over blijft voor innovatie. Dit alles bij elkaar maakt een continuous integration omgeving nog belangrijker.

## 4.2 Bestaande testautomatisering

IceMobile is, bij de start van dit project, nog niet goed bekend met testautomatisering. Er wordt geëxperimenteerd met frameworks die een aantal vooraf opgestelde unit-tests in enkele seconden tot minuten kan uitvoeren. Deze vorm van automatisering moet later ook gebruikt gaan worden voor de front-end. Daarnaast moet dit ook verder worden geautomatiseerd en gecentraliseerd worden.

Tools die worden gebruikt zijn onder andere Yadda, Mocha en SuperTest. Het opzetten van een test-tool met behulp van deze tools kost relatief weinig tijd en levert waarschijnlijk veel tijdswinst op. Met deze opstelling worden straks waarschijnlijk alle API's van IceMobile getest.

Yadda is een framework waarmee unit testen kunnen worden beschreven met normale taal en vervolgens los kunnen worden geïmplementeerd ('Afbeelding 4-2 Scenariobeschrijving' en 'Afbeelding 4-3 Implementatie scenario'). Mocha is een framework, dat wordt gebruikt door Yadda, die er voor zorgt dat deze testen ook daadwerkelijk worden uitgevoerd en daarna voor de gebruiker een rapport opstelt met de resultaten.

```
Feature: 100 Green Bottles

Scenario: Should fall from the wall

   Given 100 green bottles are standing on the wall
   When 1 green bottle accidentally falls
   Then there are 99 green bottles standing on the wall
```

Afbeelding 4-2 Scenariobeschrijving

```
module.exports = (function() {
  return English.library()
    .given("$NUM green bottles are standing on the wall", function(number, next) {
       wall = new Wall(number);
       next();
    })
    .when("$NUM green bottle accidentally falls", function(number, next) {
       wall.fall(number);
       next();
    })
    .then("there are $NUM green bottles standing on the wall", function(number, next) {
       assert.equal(number, wall.bottles);
       next();
    });
})();
```

Afbeelding 4-3 Implementatie scenario

## 4.3 Handmatig testen

Naast een stukje testautomatisering wordt er veel handmatig getest. De laatste source-code/builds worden op het eind van een sprint vaak pas compleet getest vanwege de tijdrovende scenario's die keer op keer moeten worden uitgevoerd bij wijzigingen in de applicatie. Op deze manier kunnen bugs vaak pas in een nieuwe sprint worden verholpen en worden de developers geremd in de laatste dagen van een sprint. Er kunnen namelijk geen grote features vlak voor de deadline worden geïmplementeerd vanwege de tijdsduur van het testen.

## 4.4 NodeJS

NodeJS is een platform dat is gebouwd op Chrome's Javascript runtime waarmee makkelijk snelle en schaalbare applicaties kunnen worden gemaakt. NodeJS is vooral erg geschikt voor netwerkapplicaties vanwege de asynchrone structuur van dit platform.

Binnen IceMobile wordt er meer en meer met NodeJS ontwikkeld. Waar voorheen veel met Java werd gemaakt valt de keuze steeds vaker, lees eigenlijk alleen nog maar, op NodeJS. NodeJS wordt inmiddels voor een groot gedeelte van de middleware gebruikt, het andere deel draait nog op Java. Met NodeJS kan snel een product worden gebouwd door middel van de vele modules die te vinden zijn voor dit platform. Dit is iets waar rekening mee gehouden moet worden omdat alle niet-mobile developers in ieder geval met NodeJS bekend zijn.

Het grootste voordeel ten opzichte van Java is dat er heel snel een applicatie kan worden ontwikkeld door de modulaire architectuur van NodeJS.

NodeJS leent zich perfect voor het koppelen van verschillende bestaande frameworks door middel van modules.

# 5 Projectaanpak

*Voor het managen van het project moet er een duidelijk plan zijn voor de aanpak. Daarnaast is het ook belangrijk hoe dat plan tot stand is gekomen en welke keuze waarom zijn gemaakt. Tot slot is er een planning opgenomen.*

## 5.1 Totstandkoming van de aanpak

Eerder staat beschreven dat er behoefte is aan testautomatisering van de front end van de mobiele applicaties van IceMobile. Heel veel meer informatie was er op dat moment nog niet. De eerste stappen waren dan ook om de opdracht in een kader te plaatsen en het project af te bakenen waar mogelijk en wenselijk.

Om aan meer informatie te komen hebben er korte interviews plaatsgevonden met meerdere stakeholders. Bijvoorbeeld de testers van IceMobile maar ook de developers, SCRUM-masters, systeembeheerders en mensen binnen IceMobile met ervaring op dit gebied.
Doordat iedereen zich in het zelfde bedrijfspand bevindt is het makkelijk om de stakeholders te bereiken. Daarnaast heerst er een cultuur binnen IceMobile waarbij je altijd een vraag kan stellen aan iemand.

De resultaten van de verschillende gesprekken met de verschillende stakeholders zijn verzameld en samengevat. De mogelijkheden zijn door de afstudeerder onderzocht en de conclusies voorgelegd aan de opdrachtgevers. Deze gesprekken vormden uiteindelijk de basis voor de backlog.

Vervolgens is er onderzocht welke opties er beschikbaar zijn om het testen van mobiele applicaties te automatiseren. Deze bevindingen zijn gedeeld met de opdrachtgevers. Door brainstorming, proof of concepts te bouwen en veel te lezen op internet (met name blogs) kon er een keuze gemaakt worden.

Met meer kennis over het onderwerp kan er nu specifieker worden nagedacht over wat de opdrachtgevers willen. Daarnaast kan er nu een betere inschatting worden gemaakt over de verschillende onderdelen die moeten worden ontwikkeld en over de tijd die dat waarschijnlijk gaat kosten.

Aangezien er op het einde van elke sprint een werkend product opgeleverd dient te worden is het van belang met de belangrijkste dingen te beginnen en dit verder uit te breiden. Het eerste onderdeel was een testaansturing koppelen met een testframework. Dit is de basis voor het project.

Bij verdere uitbreiding zal elke keer worden bekeken hoe dit op de beste manier in de huidige continuous integration omgeving past en wat er al aanwezig is bij IceMobile.

Tot slot moet het opgeleverde testplatform in beheer komen van iemand zodat iemand het kan onderhouden en eventueel door ontwikkelen.

## 5.2 Methoden & technieken

Gedurende het gehele project wordt er gewerkt aan de hand van afspraken. Daarnaast wordt er gebruik gemaakt van een softwareontwikkelmethode. De gekozen methode heeft veel weg van SCRUM. Het project is gestart met één persoon maar gedurende het project zijn daar telkens meer personen aan gaan deelnemen. De methoden zijn daarom ook licht gewijzigd tijdens het project. Zo is er halverwege het project een tweede persoon bijgekomen die testen is gaan implementeren en een derde persoon als projectmanager. Tegen het einde van het project zijn er meerdere stakeholders direct bij het project betrokken geraakt. Dit kwam onder andere vanwege overdrachten, doordat het systeem complexer werd en bestaande systemen waarmee gecommuniceerd moest gaan worden.

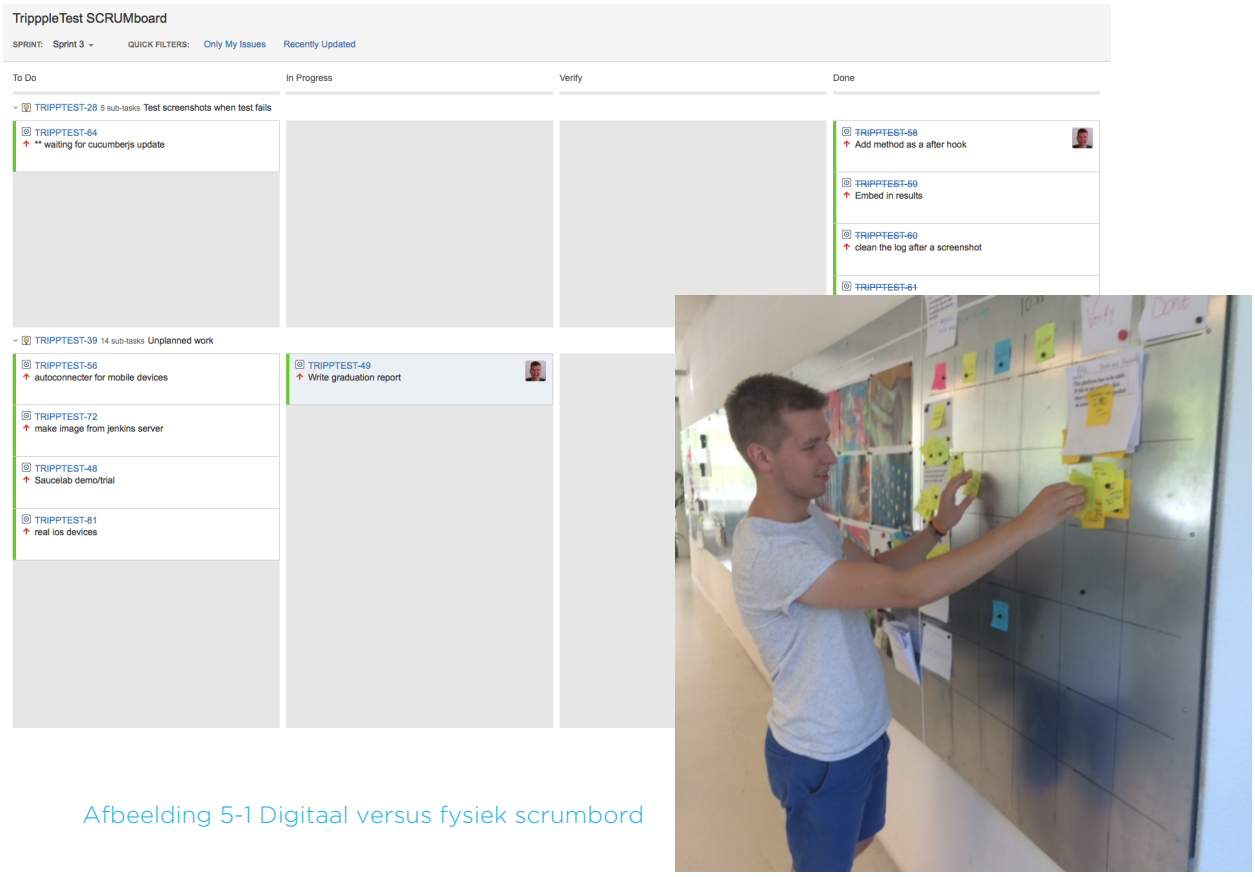
Naast de softwareontwikkelmethode is er ook met verschillende tools en technieken gewerkt. Deze worden óók kort toegelicht in de volgende sub-paragrafen.

### 5.2.1 Project

Bij IceMobile wordt gewerkt met scrum. Mij werd scrum opgelegd als softwareontwikkelmethode. Scrummen met één persoon is lastig en veel zaken worden overbodig of werken minder goed. Na dit aangegeven te hebben bij de opdrachtgever werd er verteld dat het zoveel mogelijk op scrum moet lijken om het gehele proces zo uniform mogelijk aan de andere projecten van IceMobile te houden.

De eerste weken waren gericht op kennis vergaren en onderzoeken. In deze weken is nog niet gescrumd. Bij de start van het daadwerkelijke ontwikkelen van het testplatform is er begonnen met sprints en is er een backlog aangelegd. Omdat het moeilijk te zeggen was hoeveel tijd de epic's in beslag zouden nemen is er voor gekozen om aan het begin van elke sprint een inschatting te maken van het aantal story-punten en ook per sprint zoveel mogelijk taken en user stories te hangen aan een epic. Er is begonnen met sprints van een week. Later zijn dit sprints van twee weken geworden. Een sprint van één week was gewoon te kort vanwege de overhead. Na deze switch is het project veel beter gaan lopen.

Het eerste gedeelte van software ontwikkeltraject hing er een fysiek scrumbord die dagelijks werd bijgewerkt. Later werd er bij gebrek aan ruimte en andere werkplekken overgeschakeld op een digitaal scrumbord (‘Afbeelding 5-1 Digitaal versus fysiek scrumbord’) met behulp van Jira. Jira is een digitale versie van het scrumbord met een aantal extra’s. Bijvoorbeeld real time burndown charts. (bijlage I en II)




Afbeelding 5-1 Digitaal versus fysiek scrumbord

Elke woensdag was er een meeting met alle stakeholders die op dat moment belangrijk waren. Daarnaast werd er zo goed als altijd een kleine demo gegeven aan het einde van een sprint.

Daarnaast heb ik samengewerkt met testers, NodeJS specialisten, iOS specialisten en systeembeheerders om het platform van zo hoog mogelijk kwaliteit te houden.

### 5.2.2 Brainstorm sessies

Na het onderzoeken in de eerste weken zijn er meerdere brainstormsessies geweest over welke features het liefst worden gezien in het testplatform. Deze sessies zijn uitgewerkt naar een aantal epic’s en user stories voor zover mogelijk. Deze zijn vervolgens geprioriteerd en in de backlog gezet (6.3 Opstellen epic’s).

### 5.2.3 Testbeschrijving en –aansturing

Er werd bij IceMobile al een gewerkt met Yadda als testaansturing. Yadda is een Behaviour Driven Development (BDD) framework. Met dit framework kunnen testen worden geschreven aan de hand van scenario's (zogeheten feature-files). Deze scenario's kunnen door iedereen worden opgesteld aangezien er geen programmeerkennis voor nodig is. Daarnaast wordt er duidelijk aan de opdrachtgever laten zien wat er precies getest gaat worden zonder dat dit vertaald moet worden naar 'managementtaal'. Met de feature-files alleen kunnen er nog geen testen worden afgespeeld dus moet iemand de implementatie van deze stappen nog schrijven. Om deze stappen te implementeren is er niet veel programmeerkennis nodig omdat er van uitgebreide libraries gebruik wordt gemaakt.

Voor de aansturing was de wens aanwezig om gebruik te gaan maken van Mocha. Met Mocha kan bijvoorbeeld een feature-file doorlopen worden en kunnen de resultaten vervolgens worden weggeschreven naar de console. Op die manier weet je precies wat er eventueel verkeerd is gegaan bij de testen. Uiteraard diende er eerst nog onderzoek te worden uitgevoerd of dit de juiste frameworks voor dit project zouden zijn.

Beide frameworks werken op NodeJS. Dit is één van de eisen vanuit de opdrachtgever. Alles moet op NodeJS draaien tenzij het een compleet black box framework of programma betreft die niet kan of hoeft te worden uitgebreid. Dit omdat NodeJS door het hele bedrijf wordt gebruikt. Alle terminologie wordt later in het document duidelijk als er mee gewerkt gaat worden.

### 5.2.4 Build software

Er zijn bij IceMobile al centrale servers die automatisch builds maken van de laatste code. Voor iOS is dit Xcode server. Voor Android wordt gebruik gemaakt van Gradle. Daarnaast zijn er een aantal applicaties die ook in Jenkins worden beheerd. Jenkins is een continuous integration platform.

### 5.2.5 Versiebeheer

Al het werk dat betrekking had op het project is op een iMac gedaan. Deze iMac stond in het bedrijfspand van IceMobile. Voor versiebeheer is gebruik gemaakt van Git. IceMobile heeft een GitLab server waar alles naartoe gestuurd kan worden.

### 5.2.6 Overige

Naast deze tools zijn er nog een heel aantal andere gebruikt. Deze komen echter pas later aan bod in dit verslag vanwege keuzes die gemaakt zijn gedurende het project. Het zijn er een groot aantal en het verhaal wordt er nu niet duidelijker door als alle tools worden opgesomd.

## 5.3 Planning

De planning is opgezet in grove lijnen voor de verschillende fasen. De daadwerkelijke planning zal aan het begin van elke sprint worden gemaakt. De planning is opgesteld voordat het project was begonnen en de afstudeerder nog weinig kennis had van de verschillende onderwerpen. Daarnaast wordt er gewerkt met een agile methode waardoor er tijdens het project snel beslissingen kunnen worden gemaakt op het gebied van planning. In totaal duurt het project 19 weken (‘Afbeelding 5-2 Beknopte faseplanning’).

| Fase | Week | Werkzaamheden |
|---|---|---|
| 0 | Vooraf | Opdracht omschrijven |
| 1 | 1 & 2 | Onderzoek en concerns |
| 2 | 3 t/m 8 | Testframework ontwikkelen (SCRUM) |
| 3 | 9 t/m 12 | Testframework passend maken voor continuous integration (SCRUM) |
| 4 | 13 & 14 | Testframework in continuous integration omgeving plaatsen (SCRUM) |
| 5 | 15 t/m 18 | Dashboard ontwikkelen (SCRUM) |
| 6 | 19 | Overdracht en afronding |

Afbeelding 5-2 Beknopte faseplanning

# 6 Vooronderzoek en afbakening

*Omdat er nog niet veel software beschikbaar is voor front-end testautomatisering moet er aan het begin veel aandacht worden besteed aan keuzes voor bepaalde oplossingen, denk bijvoorbeeld aan te gebruiken frameworks, de voor- en nadelen daarvan, compatibiliteit met huidige systemen. Deze fase is bedoeld om de teamleden zo goed mogelijk voor te bereiden op de softwareontwikkelingsfase. Het doel van deze fase is kennis vergaren om keuzes te maken en epic's op te stellen.*

## 6.1 Concerns en eisen op voorhand

Bij softwareontwikkeling zijn bijna altijd randvoorwaarden aanwezig. Naast de randvoorwaarden zijn er aan het begin meestal ook concerns bij de opdrachtgever. De randvoorwaarden en concerns die vroeg tijdens het project al bekend waren zijn in deze paragraaf beschreven.

In de eerste week vond de eerste vergadering plaats waaruit de volgende punten naar voren kwamen:

- Alle te gemaakte software moet in één programmeertaal worden geschreven, bij zeer hoge voorkeur NodeJS. NodeJS wordt door het hele bedrijf gebruikt.
- De testers geven aan nu met Yadda en Mocha te werken en willen dit graag terug zien in het te bouwen testframework van dit project. Het belangrijkste is dat de testers slechts met één platform willen werken.
- De tester wil een rapport kunnen inzien met de geslaagde en gefaalde tests.
- Het testframework moet calls kunnen maken op de back end om alle functionaliteit in de app te kunnen testen (bijvoorbeeld push-berichten triggeren).
- Nieuwe builds moeten automatisch worden getest.
- Als er op een knop geklikt moet worden, bijvoorbeeld 'Mijn boodschappenlijstje', dan moet dat ook op die manier kunnen worden beschreven in het testframework en niet de locatie ervan, bijvoorbeeld x=239px en y=367px.
- Er moet een mogelijkheid komen om screenshots te maken en te vergelijken met gewenste screenshots die voor de tests al beschikbaar zijn.

- De tester geeft aan dat er rekening moet worden gehouden met eventuele timing problemen in de apps. Het zou kunnen dat een app even blijft hangen tijdens het laden en het testframework daarom een false-negative teruggeeft omdat de app nog niet geheel beschikbaar is tijdens deze freeze.
- De labels van de elementen hebben bij de Android apps andere namen dan bij de iOS apps.
- Bij verschillende soorten testframeworks moet er een aparte build worden gemaakt. Er moet dan een stuk van het framework worden opgenomen in de test builds waardoor de daadwerkelijk te gebruiken build niet wordt getest. Het liefst zien de testers een oplossing waarbij dit niet het geval is.

Een gemixte lijst van concerns, eisen en wensen legden een fundering voor de afbakening van het project, het vooronderzoek en de backlog.

## 6.2 Mogelijkheden voor testautomatisering

Er zijn verschillende manieren om de testautomatisering aan te pakken. Bij aanvang werd er in plaats van testautomatisering alleen gesproken over Calabash (‘Afbeelding 6-1 Calabash’). Dit was dan ook het eerste onderwerp waar naar werd gezocht.


Afbeelding 6-1 Calabash

Calabash is open-source en gratis. Daarnaast wordt dit project ondersteund en doorontwikkeld door Xamarin. Een groot voordeel is natuurlijk dat dit project door een bedrijf met winstoogmerk wordt ontwikkeld. Met Calabash kunnen acceptatietesten automatisch worden uitgevoerd op iOS- en Android apparaten.

De interactie met de apps verloopt via een webserver. Dit is dan ook de software die wordt toegevoegd aan de app die getest wordt. Door middel van deze extra software laag kan Calabash een aantal vormen van interactie simuleren in de te testen app.

Elke actie is één van de volgende: gestures (bijvoorbeeld: rotating, swipen, pinchen), assertions (bijvoorbeeld: zoek een element genaamd ‘titel’ en kijk of dit element het woord ‘welkom’ bevat) of een screenshot (maak een screenshot en sla deze op).

Calabash maakt gebruikt van Cucumber. Met Cucumber kan gedrag worden beschreven op een manier die begrijpelijk is voor het management en andere niet-technische mensen.

De belangrijkste voordelen van Calabash:

- Er is standaard ondersteuning voor alle basis events en gestures.
- Xamarin gebruikt en ontwikkelt dit project voor zijn eigen gebruik.
- Het platform is speciaal en alleen ontwikkeld voor acceptatietesten voor mobiele applicaties.

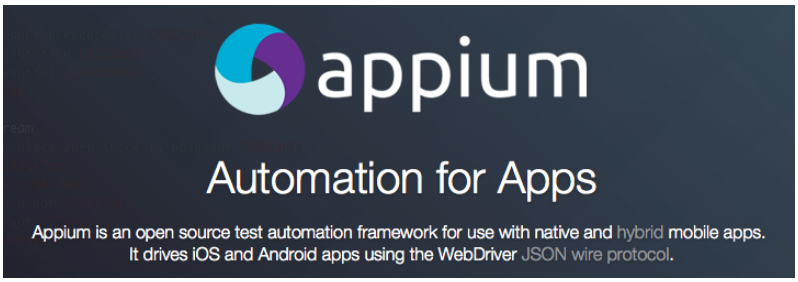
De belangrijkste nadelen van Calabash:

- Alles werkt met Ruby. Dit is ene taal die niet bekend is bij de programmeurs van IceMobile.
- De sourcecode van het te testen programma is nodig om de namen van de verschillende elementen te achterhalen.
- Er moet extra code worden toegevoegd aan de bestaande sourcecode van de te testen mobiele apps.
- Cucumber is verplicht te gebruiken bij dit platform. Dus geen Yadda.

Onder andere de standaard meegeleverde tools van Apple en Google zijn nader bekeken. Een groot nadeel van deze tools is dat het toolS zijn, in het meervoud. Wat betekent dat er dus twee keer testen moeten worden geschreven.

Zucchini is ook een framework die aan veel eisen en wensen voldeed. Met Zucchini kan er bijvoorbeeld standaard al op screenshotniveau worden getest en er is geen extra software nodig in het project van de te testen app. Daarnaast is het zeer makkelijk om met Zucchini testen te schrijven. Echter is de community niet erg actief, staat er geen groot bedrijf achter dit framework en moeten de testen worden gemaakt in Ruby en met Cucumber. Daarnaast biedt dit framework geen ondersteuning voor Android. Zucchini is daarom niet het framework dat op dit moment geschikt is voor IceMobile.

Naast een aantal testframeworks die allemaal niet voldeden aan de eisen en wensen van de opdrachtgevers voldeed Appium (‘Afbeelding 6-2 Appium’) wel.




Afbeelding 6-2 Appium

Appium werkt met hetzelfde principe als webdriver. Webdriver is een programma waarmee er voor websites acceptatietesten kunnen worden geschreven.

Net als Calabash kunnen er met Appium acties worden uitgevoerd in een app. Echter werkt dit op een andere manier dan bij Calabash. Appium maakt namelijk gebruik van Apple’s UIAutomation en Android’s UIAutomator framework. Met behulp van deze frameworks kunnen apps zonder extra code alsnog getest worden. Dit komt omdat UIAutomation automatisch aanwezig is in alle apps. Appium is, net als Calabash, een webserver die commando’s ontvangt. Maar waar Appium verschilt van Calabash is dat Appium dit op serverniveau doet en niet op de smartphone zelf. Door middel van een simpel webrequest op de Appium server maakt, en verstuurt, Appium vervolgens de UIAutomation of UIAutomator scripts naar het apparaat met de te testen app. Het resultaat wat terugkomt van de app wordt ook weer door Appium verwerkt en doorgestuurd naar de client.

Op deze manier is Appium niet afhankelijk van één taal maar kunnen er in verschillende programmeertalen libraries worden gemaakt voor Appium. Zo is er ook voor NodeJS een library, WD.js.

Het nadeel, of voordeel, van Appium is dat het geen alles-in-1 oplossing is. Om mooie en duidelijke tests te schrijven moet er bijvoorbeeld nog een testaansturing bij komen. Ook de testresultaten moeten netjes verwerkt worden na een voltooide test.

De community is ook erg actief. Er wordt veel gewerkt aan Appium door veel verschillende mensen en ook Appium wordt, net als Calabash, ondersteund en doorontwikkeld door een bedrijf. Namelijk Sauce Labs.

Er is nog één nadeel op dit moment, Appium is namelijk nog niet af. Het platform is nog niet stabiel en er moeten nog een flink aantal functies

worden geïmplementeerd. Dit alles komt ook doordat soortgelijke platformen ook nog niet volwassen zijn. Een groot aantal bugs komen indirect ook door UIAutomation van Apple. UIAutomation is ook nog lang niet perfect. iOS8 en Xcode6 moeten hier verandering in gaan brengen. En het lijkt ook dat het slechts een kwestie van tijd is voordat Appium een volwassen platform is. De community is erg actief en Appium evolueert snel.

De keuze van de afstudeerder en opdrachtgevers is gevallen op Appium in combinatie met frameworks en oplossingen op maat. Op deze manier kan alles naar eigen smaak worden ingericht.

## 6.3 Opstellen epic's

Nu er meer bekend is over de aanpak en de verschillende soorten frameworks kan er een lijst worden opgesteld met wensen en kan het project worden afgebakend.

In overleg met de opdrachtgevers zijn er een vijftal epic's opgesteld. De beschrijvingen per epic staan erbij. De epic's zijn geprioriteerd waarbij 1 de hoogste prioriteit heeft en 5 de laagste. Deze zijn in het Engels opgenomen.

De epic's zijn in een meeting met de opdrachtgevers en een expert opgesteld. Er is toen gesproken over de haalbaarheid van bepaalde taken en over de hoeveelheid van de taken. De epic's zijn opgesteld met de kennis die uit het vooronderzoek is gehaald.

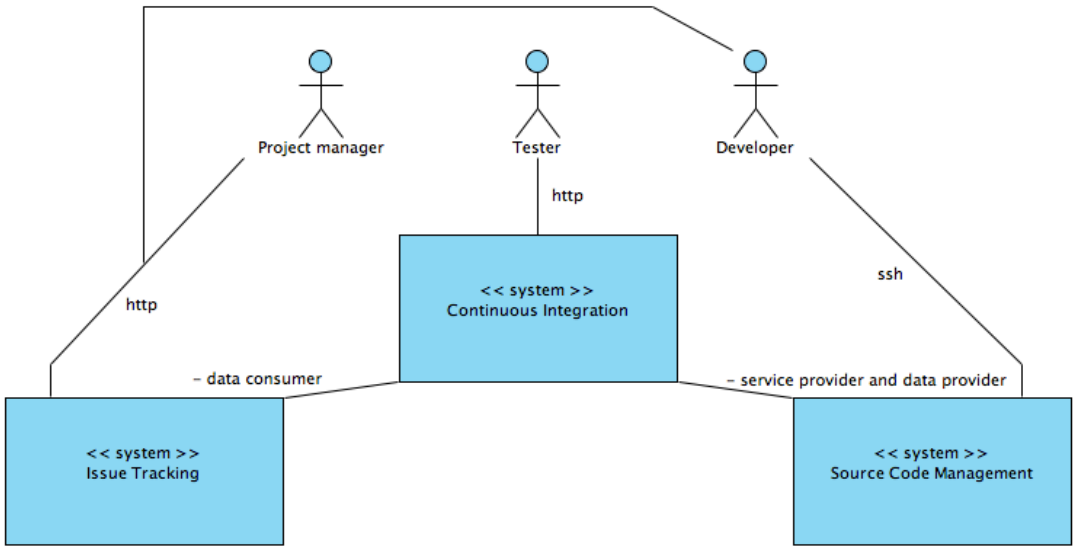
De epic's zijn terug te vinden in bijlage III.

# 7 Systeem overzicht

*Het te bouwen systeem bestaat uit een aantal verschillende subsystemen en heeft daarnaast een aantal afhankelijkheden. Dit hoofdstuk beschrijft de context van het systeem en is bedoeld om een beter beeld te krijgen van het ontwikkelingstraject dat wordt beschreven in de volgende hoofdstukken. Daar wordt er ingezoomd op de verschillende subsystemen en worden de speerpunten naar voren gehaald en onderliggende keuzes belicht.*

## 7.1 Context model

'Afbeelding 7-1 Context CI-server' geeft een algemeen beeld van de systemen, subsystemen, externe entiteiten en interfaces van dit systeem. Dit diagram is goed leesbaar voor de verschillende stakeholders aangezien er in dit hoofdstuk niet wordt ingegaan op de interne werking van de systemen, alleen de verantwoordelijkheden. Dit hoofdstuk is puur om de samenhang en het algemene plaatje te verduidelijken. In de volgende paragrafen wordt het diagram verder toegelicht.




Afbeelding 7-1 Context CI-server

## 7.2 Verantwoordelijkheden

Elke actor en elk systeem heeft zijn eigen verantwoordelijkheid. Deze paragraaf beschrijft de verantwoordelijkheden van deze objecten.

- **Actoren**
  De actoren staan symbool voor functies die kunnen worden uitgevoerd door een groep medewerkers binnen IceMobile.
  - **Developer**
    Het CI-systeem is verantwoordelijk voor de communicatie met de developer omtrent het aanbieden van gevonden bugs via de externe entiteit 'Issue Tracking system'. Daarnaast moet de developer via het Source Code Management (SCM) systeem nieuwe code kunnen opsturen die vervolgens automatisch wordt getest door het CI-systeem.
  - **Tester**
    Het CI-systeem is verantwoordelijk voor het aanbieden van een omgeving waarin de tester testscenario's kan schrijven en de testscenario's kan implementeren. Daarnaast is het CI-systeem verantwoordelijk om de testrapporten van de geautomatiseerde testen aan te bieden. Tot slot moet een functie aanbieden waarmee de tester een gefaald scenario in Jira moet kunnen zetten door middel van een simpele muisklik.
  - **Project manager**
    Het CI-systeem is verantwoordelijk voor het aanbieden van nieuwe issues (taken) voor het 'Issue Tracking systeem' waardoor het 'Issue Tracking systeem' deze kan aanbieden aan de project managers.
- **Issue Tracking systeem**
  Het CI-systeem is niet verantwoordelijk voor het beheren en verwerken van nieuwe taken die voortvloeien uit een gefaalde test. Daar is het Issue Tracking systeem verantwoordelijk voor. Het CI-systeem is wél verantwoordelijk voor het aanbieden van een nieuwe taak wanneer een tester dat gewenst acht.
- **Source Code Management systeem**
  Het CI-systeem is niet verantwoordelijk voor het beheren van de source code binnen IceMobile en het initiëren van het ophalen van nieuwe source code. Het CI-systeem is wél verantwoordelijk voor het ophalen van de correcte, meestal nieuwe, source code.

## 7.3 Interfaces

Het CI-systeem maakt gebruik van en bezit een aantal interfaces. De verantwoordelijkheden van de interfaces zijn in 'Afbeelding 7-2 Interfaces CI-server' beschreven.

| Van | Naar | Verantwoordelijkheid | Eigenaar |
|---|---|---|---|
| Issue Tracking | CI Systeem | Het issue tracking systeem biedt een interface aan waarmee het CI-systeem een issue in het issue tracking systeem kan plaatsen. | Atlassian |
| Source Code Management | CI Systeem | Het SCM-systeem biedt een interface aan waarmee de juiste source code kan worden opgehaald. | GitHub en GitLab (intern) |
| CI Systeem | Source Code Management | Het CI-systeem biedt een interface aan waarmee het SCM-systeem het CI-systeem kan melden dat er nieuwe source code beschikbaar is. | CI-systeem |

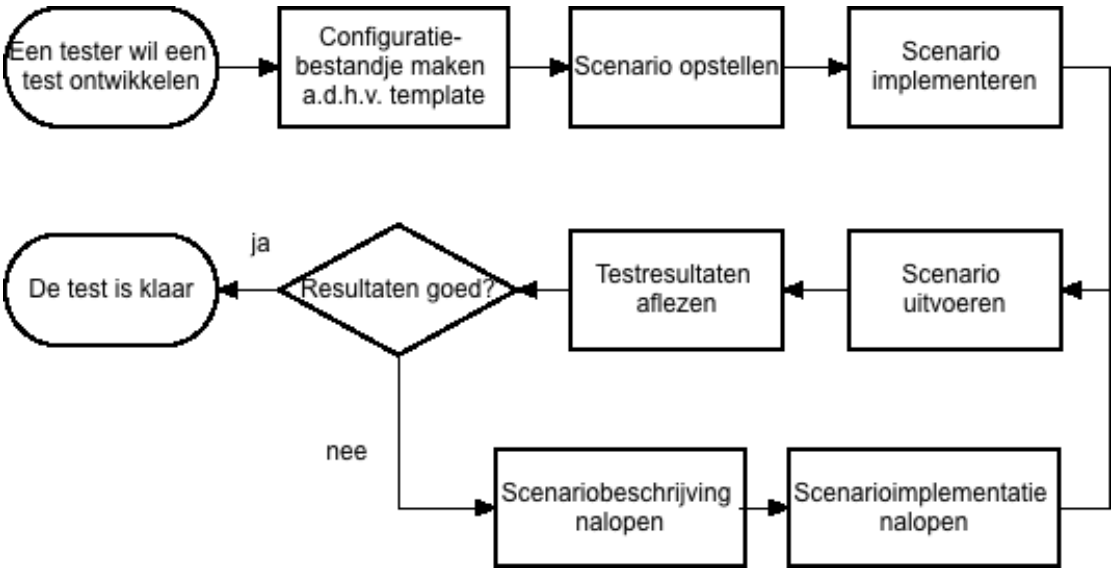
Afbeelding 7-2 Interfaces CI-server

# 8 Testframework

*De eerste stap op weg naar een geautomatiseerd systeem gebaseerd op Appium is: Appium plaatsen in een testframework. Hiervoor is dus een testaansturing nodig. Daarnaast moet dit te ontwikkelen framework er voor zorgen dat Appium's stabiliteitsproblemen worden opgelost. De opdrachtgever gaf aan graag Yadda en Mocha te zien voor de testbeschrijvingen en –aansturingen aangezien zij daar zelf ook mee werken.*

## 8.1 Flow chart testontwikkeling

Een tester moet een aantal dingen doen voordat er een geautomatiseerde test aanwezig is. Alle grote testframeworks werken op een soortgelijke manier. Namelijk eerst scenario's schrijven en daarna de scenario's implementeren. Op basis daarvan ontstaat 'Afbeelding 8-1 Flow chart testontwikkeling'.




Afbeelding 8-1 Flow chart testontwikkeling

## 8.2 Keuze frameworks

In het vooronderzoek is Appium als meest geschikt bevonden voor dit project. Echter is Appium slechts een deel van het gehele testproces. Om de testen te beschrijven, te implementeren en de resultaten in te kunnen zien is er nog andere software benodigd.

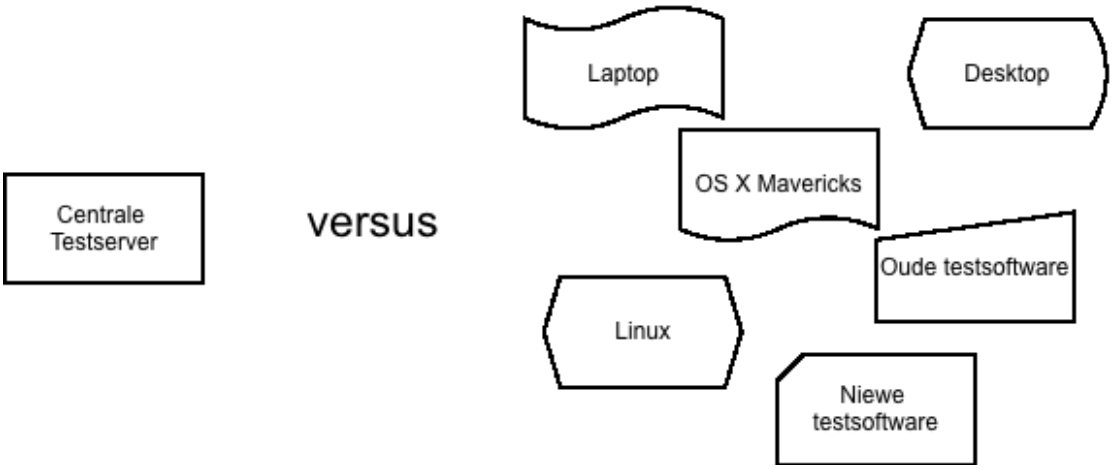
### 8.2.1 Nieuw of bestaand

Deze software kan zelf worden geschreven of er kan worden gekozen om bestaande frameworks hier voor te gebruiken. Beide hebben voordelen en

nadelen. Na wat speurwerk op internet, en de wensen van de opdrachtgevers in het achterhoofd, werd al snel duidelijk dat er een aantal frameworks te vinden zijn die zeer geschikt zijn om de testen te beschrijven en aan te sturen. Yadda is een grote wens vanuit de opdrachtgever en ziet er, buiten de niet zeer actieve community, goed uit qua functionaliteit. Dit is dan ook direct een valkuil om met Yadda verder te gaan. Aan de andere kant bespaart het de testers nu veel tijd als Yadda gebruikt wordt. De testers zijn er namelijk al mee bekend.

### 8.2.2 Grote voordeel van Yadda is ...

Yadda is gebaseerd op CucumberJS. CucumberJS is net als Yadda een platform waar testen voor kunnen worden geschreven aan de hand van scenario's. Het verschil met Yadda zit hem in de veel actievere community en een andere vorm van aansturing. Zo bezit Yadda standaard functionaliteit waarmee testen kunnen worden aangestuurd met een aantal verschillende frameworks. Dit is natuurlijk erg handig als er al een CI-omgeving is die werkt met een bepaalde testaansturing. Echter is er bij IceMobile alleen nog maar sprake van decentrale testautomatisering, dat wil zeggen dat de testers de testen lokaal op hun eigen machine draaien en niet op een centrale server waar alle testen bij elkaar staan opgeslagen ('Afbeelding 8-2 Centraal versus decentraal').




Afbeelding 8-2 Centraal versus decentraal

### 8.2.3 Twee categorieën

Naast Yadda en CucumberJS zijn ook Jasmine, Vows en Mocha naar voren gekomen tijdens het zoeken naar de juiste frameworks. Al snel viel op dat deze vijf test frameworks op te delen zijn in twee groepen. Namelijk Yadda en CucumberJS in één groep en de andere drie (Jasmine, Vows en Mocha) in de andere.

Yadda en CucumberJS zijn twee frameworks die volledig gebaseerd zijn op BDD. BDD staat voor Behaviour Driven Development. Met BDD kan er duidelijk door elk persoon een testscenario worden opgesteld. Er wordt namelijk op gedrag getest. Een voorbeeld:

1. Gegeven, de zegeltjes applicatie is opgestart
2. Als ik door het menu naar beneden scroll
3. en ik op 'help' druk
4. Dan ben ik op de pagina met FAQ.

De testen zijn leesbaar voor iedereen, zelfs de klanten. De andere drie frameworks zijn prima frameworks om mee te testen maar hebben geen aparte testscenario beschrijvingen (zoals in het voorbeeld) en vallen daarom af. De testen zijn slechter leesbaar of misschien zelfs helemaal niet voor niet-technische mensen.

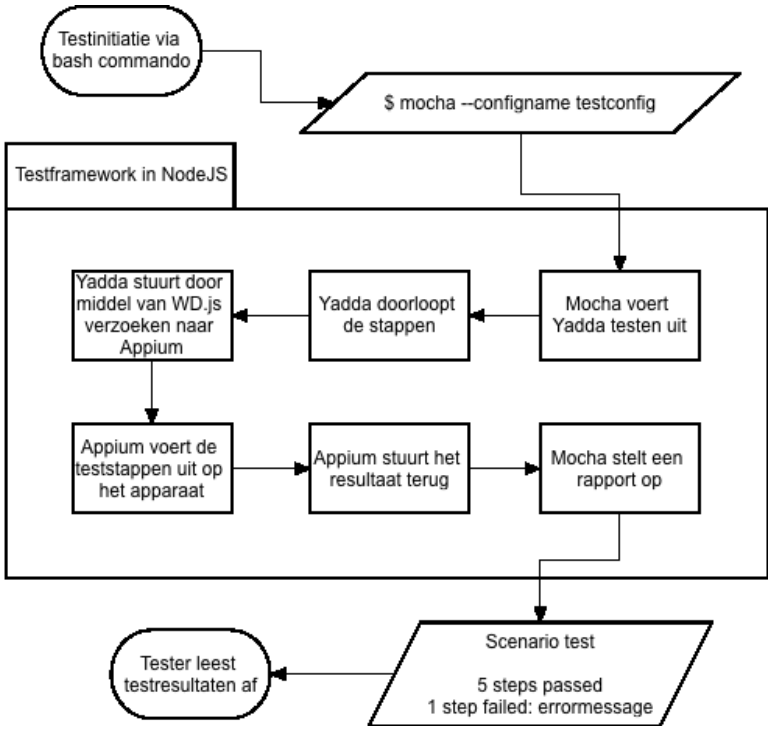
### 8.2.4 Uiteindelijke keuze

Yadda of CucumberJS zijn de twee opties die het meest geschikt zijn voor IceMobile en dit project op dit moment. CucumberJS biedt misschien iets meer perspectief in de toekomst vanwege de grotere community en iets simpeler ogende configuratie. Aan de andere kant is Yadda goed te plaatsen in verschillende omgevingen en kan dit in de verdere toekomst misschien veel problemen voorkomen.

De opdrachtgever had minder twijfel en wilde graag Yadda gebruiken vanwege ervaring en een aantal bestaande testen die al geïmplementeerd zijn met Yadda. Dit was dan ook de doorslaggevende factor bij het maken van de uiteindelijke keuze.

## 8.3 Samenwerking frameworks

Nu de eerste keuzes zijn gemaakt kan er gestart worden met de ontwikkeling van de kern van de gehele CI-omgeving, het testframework. Om te communiceren met Appium wordt er gebruik gemaakt van WD.js. Dit is niet meer dan een client-library waarmee opdrachten kunnen worden verstuurd naar Appium vanuit een NodeJS omgeving. WD.js wordt aangeraden op de Appium website en heeft een steeds rijkere community naarmate Appium groeit. De eerste schets van de samenwerking tussen de verschillende frameworks is te zien in 'Afbeelding 8-3 Samenwerking frameworks'.




Afbeelding 8-3 Samenwerking frameworks

Omdat het hele testframework zich in één project bevindt kunnen er snel en makkelijk stukken code worden geschreven om het framework te koppelen. Dit is de kracht van NodeJS.

## 8.4 Werkwijze

Het product dat in deze fase wordt ontwikkeld zal uiteindelijk de kern van het test framework worden. Het is daarom van belang dat dit 'hart' van het hele framework stabiel is en geen false positives/negatives rapporteert. Om dit doel te bereiken is er gekozen om stap voor stap dit product op te bouwen. De eerste stap is daarom ook om Yadda (in combinatie met Mocha) werkend en stabiel te krijgen met Appium.

## 8.5 Ontwikkelen

Na een kleine test voor Yadda geschreven te hebben moesten de stappen worden geïmplementeerd. Om deze stappen te implementeren moet Appium worden gekoppeld aan het project. Appium is namelijk een webserver die nu nog los staat van het project.

### 8.5.1 Verbinden met Appium

Om grote fouten te voorkomen wordt de Appium app gebruikt van de website. Dit is een losse applicatie met een GUI er overheen, duidelijk en simpel. De Appium server kan worden gestart met een druk op de knop

waarna er commando's naar toe kunnen worden gestuurd. Dit kan aan de hand van het JSON-wire protocol. WD.js is nog volop in ontwikkeling maar beschikt al over veel mappings voor Appium. Appium is nog erg nieuw in deze fase van het project, het is zelfs nog in bèta en bezit nog niet alle functies die benodigd zijn om een mobiele app helemaal te kunnen testen. Zo mist er ook nog de benodigde documentatie om met WD.js verbinding te maken met Appium.

Praktisch alle documentatie en voorbeelden voor Appium ging in deze fase van het project over de Java client. Doordat er nog weinig te vinden was op de verschillende fora nam dit veel tijd in beslag. Daarnaast is Appium nog verre van productie-stabiel. Met het vergelijken van verschillende oplossingen voor normale webdrivers in NodeJS en voor Appium in andere talen kon er uiteindelijk een verbinding worden gelegd met de Appium server ('Afbeelding 8-4 Codevoorbeeld verbinding met Appium').

```
var config = {
	platformName: 'iOS',
	platformVersion: '7.1',
	browserName: 'Safari',
	deviceName: 'iPhone Simulator',
	newCommandTimeout: 3600,
	app: '../../app/testApp.app',
	browserName: 'Safari',
	deviceName: 'iPhone Simulator'
};

var driver;

before(function(done) {
	// Configure the server
	driver = wd.promiseChainRemote('localhost', 4723);
	// Let's init our app.
	driver.init(config)
		.nodeify(done);
});

after(function(done) {
	// Tell the webdriver (appium in our case) to quit
	browser.quit()
		.nodeify(done);
});
```

Afbeelding 8-4 Codevoorbeeld verbinding met Appium

Ondanks de simpele code heeft het relatief veel tijd gekost om dit werkend te krijgen door de eerdergenoemde problemen. Dit is iets wat door het hele project terugkomt. Nieuwe software met weinig documentatie kost simpelweg soms extra tijd.

### 8.5.2 Eerste test voor mobiel

Met een verbinding konden de eerste tests worden uitgevoerd met Appium. De eerste tests waren puur voor iOS. Dit had ook Android kunnen zijn maar de simulator van iOS zit veel beter in elkaar dan de emulators van Android en daarom viel de eerste keuze op iOS.

De eerste tests werden uitgevoerd op een simpele demo-applicatie die bestond uit een tekstveld, een knop en een tweede scherm achter de knop. Met behulp van de accessibility inspector van Xcode of Appium kon de naam van het juiste element worden opgezocht om deze vervolgens te gebruiken in de testimplementatie.

### 8.5.3 Stabiliteitsproblemen

Eén van de eerste dingen die opviel was de stabiliteit van Appium. Appium crashte vaak en was dan niet meer bereikbaar. Niet alleen door crashes werd Appium onbereikbaar, ook verlopen sessies of niet opgeruimde sessies gaven veel problemen. Vooral bij een wat langere inactieve periode wilde Appium niet altijd een nieuwe sessie starten en moest de server handmatig opnieuw worden gestart.

Snel werd duidelijk dat het fixen van deze problemen de hoogste prioriteit had. Als deze problemen niet kunnen worden verholpen moet er namelijk alsnog naar een alternatief worden gezocht of moet er eerst worden gestart met andere epic's in de hoop dat Appium snel een stabiel platform wordt, dit zou een enorm risico met zich meebrengen.

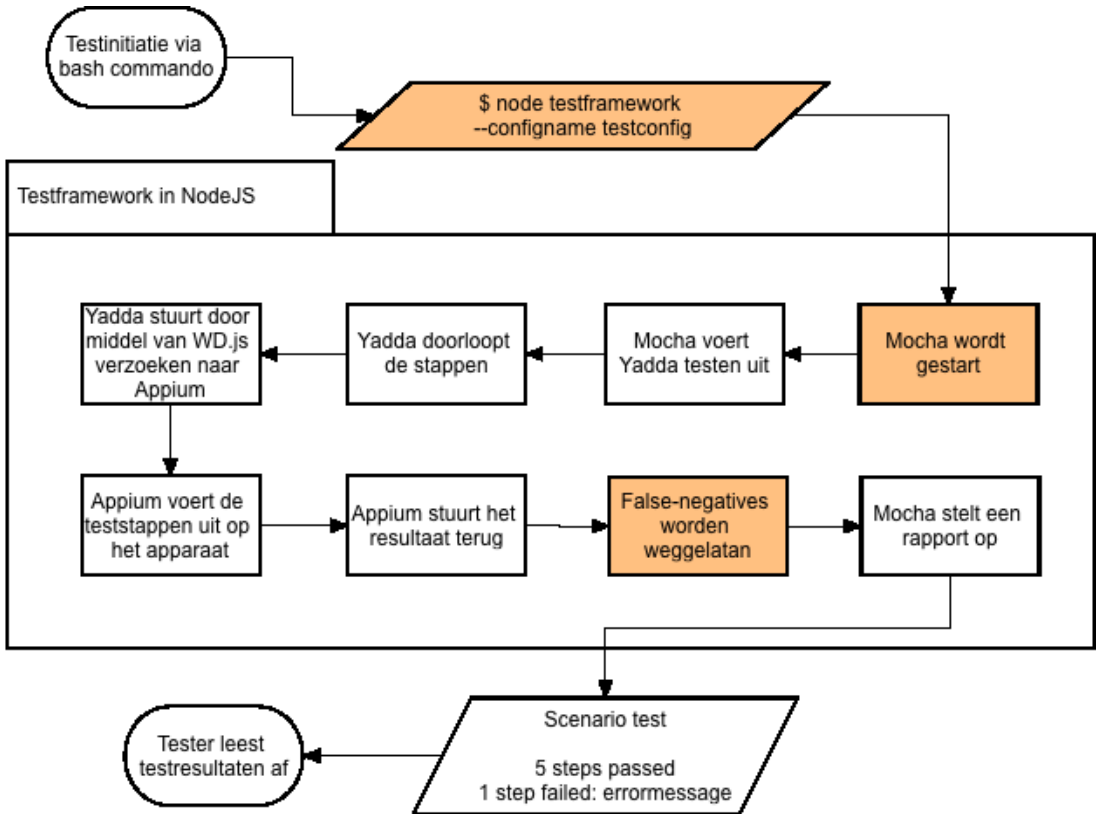
#### 8.5.3.1 Crashen van Appium

De kant-en-klare Appium app is vervangen door de NodeJS module en werd opgenomen als dependency in het project. Appium kon nu gestart worden met een commando via de terminal. Om het crash-probleem op te lossen wordt Appium vanaf nu gestart met behulp van de NodeJS module Forever. Met Forever kan een NodeJS applicatie worden gemonitord. Als de applicatie crasht start Forever de applicatie weer op en blijft daarmee continu draaien. Vanwege de architectuur van NodeJS kon dit snel worden opgelost.

#### 8.5.3.2 Sessiefouten Appium

Het probleem met de sessies ligt wat lastiger. Omdat een geweigerde verbinding niet direct zorgt voor een crash van Appium en het niet helpt om een nieuwe sessie te starten omdat zich dan precies hetzelfde probleem voor doet als eerst moet de output van Appium worden uitgelezen en worden verwerkt.

Om die reden is er een nieuwe aansturing gemaakt boven op de Mocha aansturing. Waar Mocha eerst direct door de tester werd aangeroepen gaat daar nu een kleine laag tussenkomen die de false-negatives eruit haalt. De flow chart van 'Afbeelding 8-3 Samenwerking frameworks' verandert dan in de volgende 'Afbeelding 8-5 Samenwerking frameworks met testrunner'.




Afbeelding 8-5 Samenwerking frameworks met testrunner

In de extra laag die Mocha aanstuurt kan er door middel van events informatie worden verzameld. Zo kan er door middel van het event 'fail' worden gekeken wanneer een test is gefaald. Het falen van testen is op zich natuurlijk niet verkeerd want dat is juist wat de testers willen weten maar bij het falen van een sessie faalt de test ook. En aangezien het verbinden met Appium altijd gebeurd in de 'before-all hook' (dit is een stap die wordt uitgevoerd voordat de testen daadwerkelijk worden gestart) kunnen alle testen die falen met de titel 'before-all hook' worden weggelaten van het testrapport. Vervolgens kan dit scenario opnieuw worden uitgevoerd en is het testrapport vervolgens goed (zie bijlage IV voor een screenshot).

### 8.5.4 Configuratie

Naast het stabiliseren van het platform was er ook een vorm van configuratie nodig. Er moet immers, zonder dat de code wordt aangepast, een aantal variabelen kunnen worden meegegeven. Denk aan bijvoorbeeld aan de te testen app, het platform (iOS of Android) en nog een aantal zaken. Binnen NodeJS wordt er veel met JSON bestanden gewerkt zo ook

Appium. Daarnaast wordt er bij websites ook veel gewerkt met JSON bestanden en aangezien er nog een dashboard gemaakt moet worden voor de CI-omgeving is er gekozen om de configuratie in een JSON bestand op te slaan.

Met behulp van de module convict kunnen er omgevingsvariabelen en parameters worden uitgelezen die de tester kan meegeven bij het starten van een test.

## 8.6 Veranderingen teamsamenstelling

Aan het eind van deze fase kreeg het project steeds meer aandacht van IceMobile. Er kwam een tester bij het project die aan de slag ging met het schrijven van scenario's en het implementeren van deze scenario's. Dit zorgde direct voor een beter product. De tester had namelijk punten die hij miste in het framework of niet fijn vond werken. Deze punten konden dan aan de backlog worden toegevoegd.

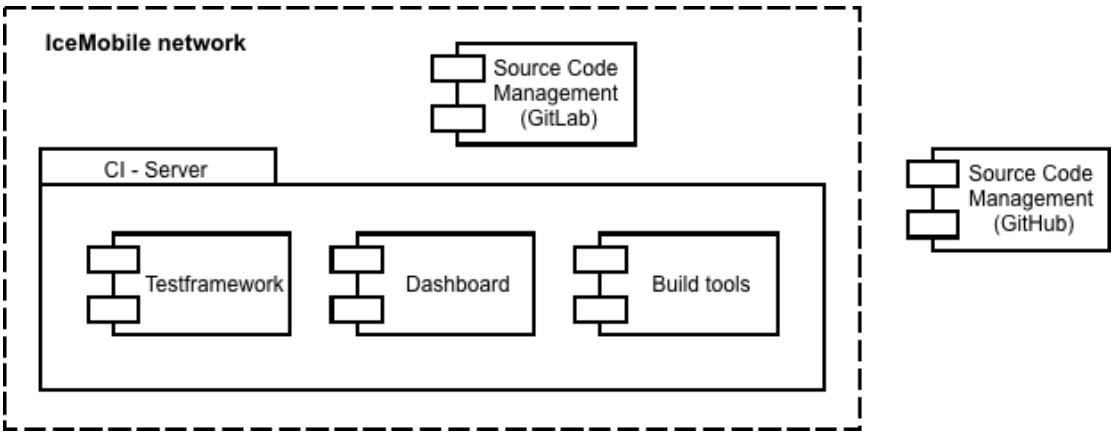
Niet alleen een tester kwam bij het team, er kwam een projectmanager bij om het project in zo goed mogelijke banen te leiden. De stand-ups kregen meer betekenis omdat er meer mensen bij het project betrokken raakten en werden nu altijd uitgevoerd, waar dit voorheen niet altijd gebeurde. Het project liep vanaf dit moment net iets strakker. Misschien is er te vroeg begonnen met scrummen omdat het project vanaf dit punt pas goed ging lopen qua softwareontwikkelmethode. Hiervoor was het vooral veel onderzoek.

# 9 Testframework in CI-omgeving plaatsen

*Een testframework is slechts een deel van de automatisering. De testen kunnen worden beschreven en worden uitgevoerd, vervolgens krijgt de tester een overzicht van testen. De volgende stap is het automatiseren van het starten van de testen en het verzamelen van de resultaten in een dashboard.*

## 9.1 Overzicht CI-server

Op de CI-server moeten een aantal subsystemen komen te staan. Namelijk de build tools om een app te bouwen van de laatste source code, het testframework om de app te testen en een dashboard om de testen te laten starten en de testresultaten weer te geven (‘Afbeelding 9-1 Subsystemen CI-server’).




Afbeelding 9-1 Subsystemen CI-server

## 9.2 Keuzes CI-omgeving

Er zijn vele manieren om dit aan te pakken. Eerst moet er een trigger zijn om de testen te starten. Dit kan bijvoorbeeld elk uur zijn of nog beter na een nieuwe build. Pas bij nieuwe source code moeten de testen opnieuw worden gestart en niet eerder.

### 9.2.1 Hoe een test initiëren?

Omdat IceMobile een eigen GitLab server heeft was dit het perfecte systeem om een test te starten. GitLab is net als GitHub een SCM voor Git repositories. Met GitLab kan er door middel van een webhook een event worden afgevuurd. Een webhook is niet meer dan een webrequest die afgevuurd kan worden op een webserver. Met een webrequest kan vervolgens de het testframework worden gestart. Echter moet de source-code eerst nog gecompileerd worden. En op het eind moeten de resultaten

ook nog worden weggeschreven in een soort van database. Kortom, veel stappen die elkaar op volgen in tijd wat in een té ingewikkeld systeem kan resulteren als er niet goed wordt nagedacht over de communicatie tussen de verschillende systemen. Daarnaast zal IceMobile niet de eerste zijn met dit probleem dus is het belangrijk om eerst verder uit te zoeken wat er allemaal beschikbaar is.

### 9.2.2 CI-platform

Teamcity, Travis, Hudson en Jenkins zijn de drie grote continuous integration platformen volgens meerdere bronnen (zie literatuurlijst). Elk van deze CI-platformen heeft zijn voor- en nadelen. Travis is zeer simpel in gebruik maar heeft als nadeel dat de servers in de cloud draaien en de code dus over het web moet gaan, Travis valt om deze reden direct af omdat IceMobile dit niet wenst.

Hudson en Jenkins lijken erg veel op elkaar. Dit komt doordat Jenkins een fork is van het Hudson project. De makers van Hudson werken nu allemaal aan de Jenkins fork, samen met vele andere open-source ontwikkelaars. Als het aantal commits en mensen binnen de community van Jenkins gedurende de laatste maanden worden vergeleken met die van Hudson valt op dat de Jenkins community veel meer leeft. Daarnaast gebruikt IceMobile zelf ook Jenkins voor een aantal zaken. Zij zijn overgestapt van Hudson naar Jenkins vanwege deze reden. Hudson valt daarom ook af.

Teamcity en Jenkins blijven dan over en zijn beide prima platformen voor een CI-omgeving. Teamcity is wellicht makkelijker op te zetten dan Jenkins. Jenkins is daarentegen open-source, volledig aanpasbaar en gratis. Als IceMobile veel .Net applicaties zou ontwikkelen zou Teamcity het winnen. Dit is echter niet zo. Sterker nog, Jenkins wordt op dit moment gebruikt voor een paar projecten en bij de keuze voor Jenkins zal dit de systeembeheerders dan ook werk besparen. Jenkins lijkt daarom een verstandige keuze aangezien beide prima platforms zijn.

## 9.3 Werkwijze

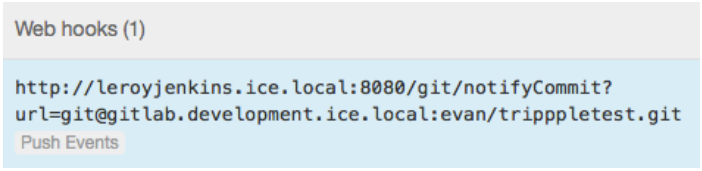
Jenkins zal gaan fungeren als een soort task runner voor de verschillende taken die moeten worden uitgevoerd. Om het CI-systeem op te bouwen is er gekozen om bij het ‘begin’ van het CI-proces te beginnen, namelijk het ontvangen van de webhook van GitLab. Gedurende paragraaf 9.4 zal er bij elke grote stap een diagram worden opgenomen die langzaam groeit ter verduidelijking voor de lezer.

## 9.4 Opzetten en ontwikkelen

Jenkins moest eerst worden opgezet en geconfigureerd. Op de website van Jenkins kan er een installatiebestand worden gedownload. Na Jenkins te hebben geïnstalleerd, wat best veel tijd koste vanwege een aantal extra handelingen die moeten worden uitgevoerd.

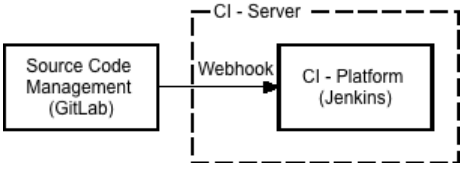
### 9.4.1 Het proces starten

Na de installatie, was de eerste stap om een taak te starten door middel van een nieuwe commit op de GitLab server van IceMobile. Hiervoor was een extra plugin nodig en kan er door middel van een webhook (‘Afbeelding 9-2 GitLab webhook’) een taak worden gestart.




Afbeelding 9-2 GitLab webhook

Na de juiste plugin en instellingen gevonden te hebben en alles werkend te hebben gekregen kon er een taak worden gemaakt in Jenkins die de source code van een bepaald project van GitLab ophaalt (‘Afbeelding 9-3 CI workflow 1’).




Afbeelding 9-3 CI workflow 1

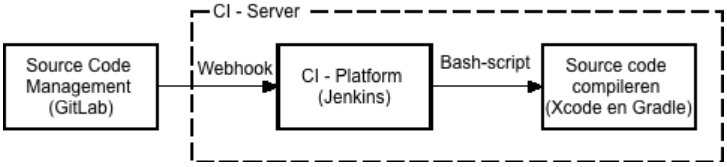
### 9.4.2 Compileren van de source code

Na het ophalen van de laatste source-code van GitLab moet het project nog worden gecompileerd. Voor iOS kan dit met de Xcode-plugin voor Jenkins. Xcode is de ontwikkelomgeving voor iOS applicaties. Xcode dient zelf ook geïnstalleerd te zijn. De Xcode-plugin roept onderwater xcodebuild aan. Dat zijn de build tools van Xcode die kunnen worden aangeroepen via de command line.

In de configuratie van xcodebuild kunnen een aantal opties worden ingesteld, bijvoorbeeld voor welke SDK de app moet worden gebouwd, welk Xcode schema moet worden gebruikt, waar de mobile provisioning file staat en waar de gecompileerde app moet worden weggeschreven. Voor een aantal dingen was er overleg nodig met het iOS team. Zo was het

onduidelijk welke schema's er worden gebruikt voor bepaalde builds en moest de CI-server toegang krijgen tot iOS ontwikkelteam om de apps daadwerkelijk te bouwen. Het lijkt allemaal niet veel werk maar er komt uiteindelijk toch meer bij kijken doordat dingen niet standaard werken.

Voor Android was het anders. Android wordt gebouwd door middel van Gradle. Dit is ook een vorm van automatisering. Met Gradle kunnen apps worden gebouwd, getest en gedeployd. Aangezien er al een aantal Gradle taken staan beschreven kan met een simpel commando de bouwtaak worden gestart. Na het bouwen hoeft de app alleen nog maar te worden verplaatst naar de goede map. Dit kan door middel van een bash-script ('Afbeelding 9-4 CI workflow 2').




Afbeelding 9-4 CI workflow 2

### 9.4.3 Testframework initiëren en test draaien

Nu de app gebouwd is, zowel voor iOS en Android, kunnen de vooraf opgestelde testen worden uitgevoerd door middel van het testframework. Als de test wordt gestart wordt de job-naam meegestuurd van Jenkins als configuratienaam. Aan de hand van de job-naam wordt vervolgens het juiste configuratiebestand opgehaald net als een lokale testuitvoering.

In het testframework moet er extra code worden toegevoegd om alles te kunnen automatiseren. Waar de app nu nog lokaal wordt uitgelezen door het test framework moet de zojuist gecompileerde app, als de test door Jenkins wordt geïnitieerd, opgehaald worden vanuit Jenkins. Appium beschikt standaard over een feature waarbij de app kan worden gedownload via internet. Hiervoor moet de app dan nog wel worden ingepakt. Deze functie zit standaard ingebakken in OS X dus dat is niet ingewikkeld.

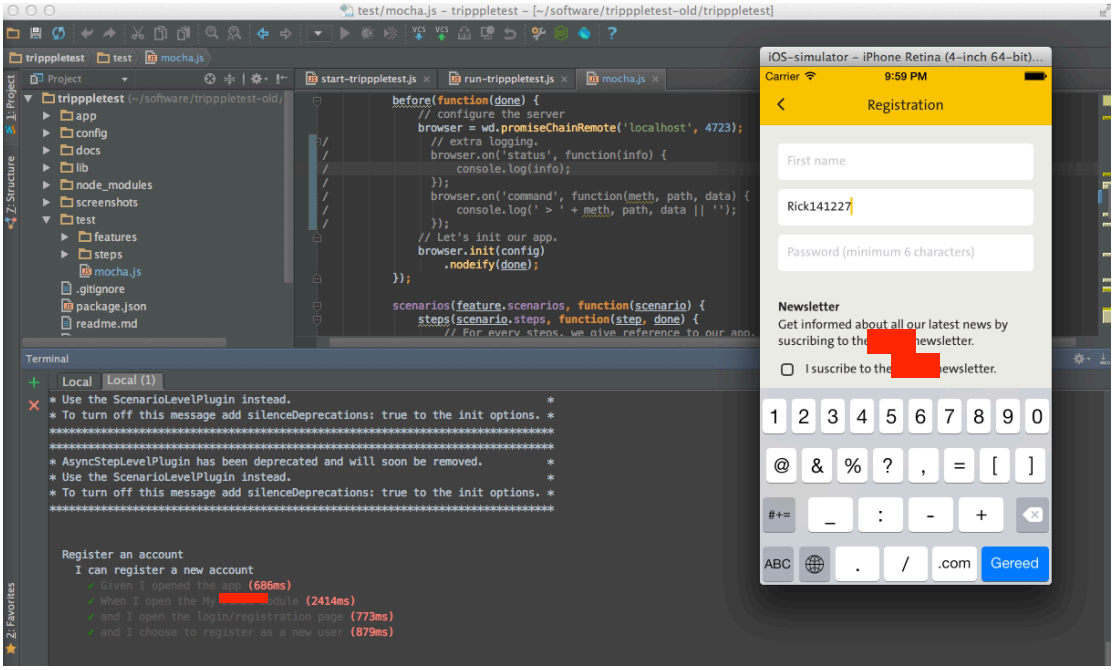
In Jenkins moet de app na het bouwen dus standaard worden gecomprimeerd. Dit gebeurt door middel van een bash-script. Daarna moet er in het testframework een hyperlink worden opgebouwd om de app te downloaden vanuit Jenkins ('Afbeelding 9-5 Opbouw hyperlink nieuwe build').

```
config.app = process.env.JOB_URL + 'ws/builds/' +
             process.env.BUILD_NUMBER + '/'+ process.env.JOB_NAME;
```

Afbeelding 9-5 Opbouw hyperlink nieuwe build

Jenkins biedt standaard een aantal omgevingsvariabelen aan die kunnen worden uitgelezen door het testframework. Met die variabelen, zoals in ‘Afbeelding 9-5 Opbouw hyperlink nieuwe build’ te zien, kan er onder andere een hyperlink worden opgebouwd.

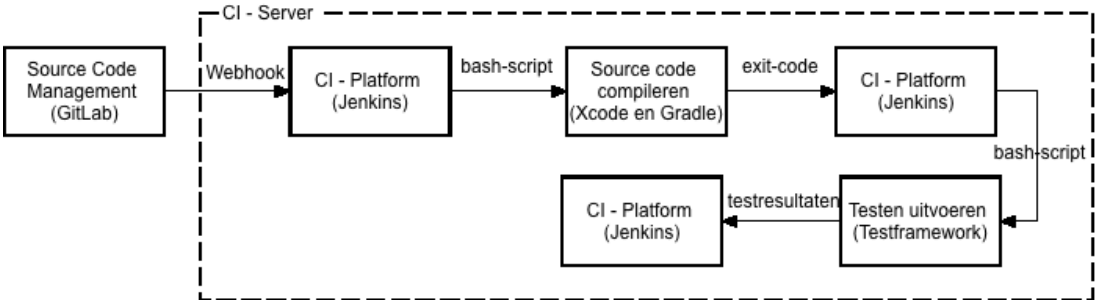
Vervolgens kan de app worden gestart door de CI-server (‘Afbeelding 9-6 Live test run’).




Afbeelding 9-6 Live test run

Nadat de testen zijn uitgevoerd moeten de resultaten nog worden verzameld in Jenkins. Normaliter zijn de resultaten gewoon in de terminal te zien maar dat is nu niet genoeg aangezien de tester er niet bij is als de testen worden uitgevoerd. Mocha heeft zelf een xunit reporter die bestanden maakt die kunnen worden uitgelezen door Jenkins. Echter worden de testen niet netjes weergegeven in Jenkins. Mocha biedt een optie aan om eigen reporters te gebruiken en op internet werd de reporter ‘xunit-file’ gevonden die wel de resultaten netjes wegschrijft in Jenkins.

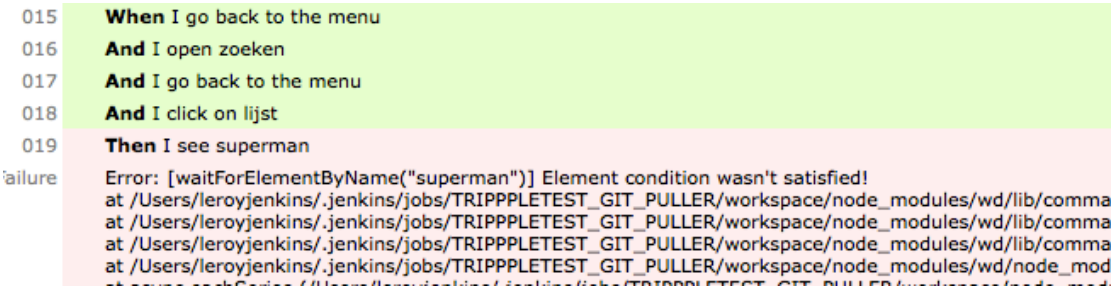
De laatste stap is het kopiëren van het testrapport naar Jenkins. Dit gebeurt lokaal (aangezien alles op de CI-server draait). Het testframework verplaatst de testresultaten naar de juiste map. (‘Afbeelding 9-7 CI workflow 3’)



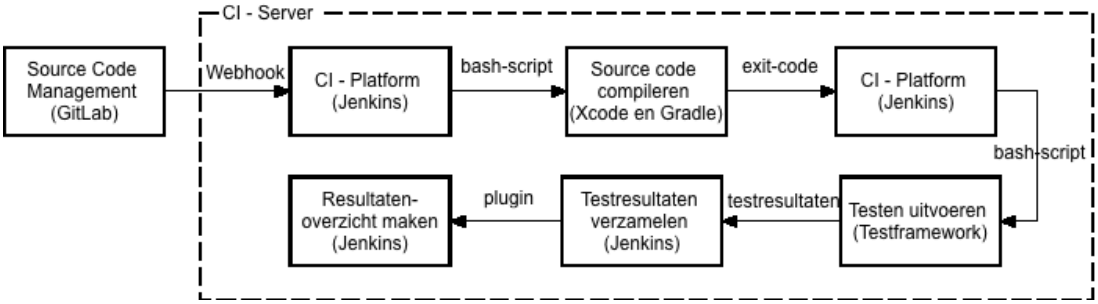

Afbeelding 9-7 CI workflow 3

### 9.4.4 Testresultaten verzamelen

Jenkins krijgt een exit-code van het testframework en kan vervolgens de testresultaten verzamelen (want die zijn door het testframework gekopieerd naar een map waar Jenkins bij kan). Vervolgens kan de tester precies zien waar de testen eventueel zijn gefaald (‘Afbeelding 9-8 Overzicht testrapport Jenkins’). (‘Afbeelding 9-9 CI workflow 4’)




Afbeelding 9-8 Overzicht testrapport Jenkins




Afbeelding 9-9 CI workflow 4

## 9.5 Problemen

Tijdens het maken van de CI-server werd er veel gebruik gemaakt van verschillende frameworks, plug-ins en platforms. Deze software is zo goed als allemaal gratis en daarnaast ook open-source. Het voordeel hiervan is dat er snel nieuwe features kunnen worden geschreven en bugs kunnen worden gefixt.

Het nadeel hiervan is dat er regelmatig bugs in de software zitten. En daarnaast is veel software nog nieuw op dit gebied. Appium heeft bijvoorbeeld wekelijks een groot aantal verbeteringen, vaak betreft stabiliteit en nieuwe features.

Doordat er regelmatig bugs in de software zitten moeten er nog wel eens workarounds worden geschreven/ontwikkeld. Dit kost veel tijd. De meeste tijd is daarom ook uitbesteed aan het oplossen van bugs en het vinden van juiste oplossing om met een aantal tekortkomingen om te gaan. Naar mate de tijd vordert komt dit minder voor en leer je de frameworks kennen waardoor dit sneller gaat.

## 9.6 Live testomgeving

Een proof of concept van de CI-omgeving leek goed te werken. De volgende stap was om dit te testen in een acceptatieomgeving. Met het opzetten van deze omgeving kwamen er een aantal problemen aan het licht.

Jenkins wordt gestart onder een ander gebruikersaccount dan de ingelogde gebruiker op dat moment. Dit leverde problemen op met omgevingsvariabelen. Uiteindelijk werd het daemon-script aangepast van Jenkins met een aantal extra omgevingsvariabelen. Dit is een soort van bash-script speciaal voor Unix systemen. Zo kon NodeJS op de juiste manier worden gebruikt maar ook de Android SDK, Android Tools, Apache Ant en Java konden nu worden gebruikt vanaf de command line in Jenkins.

Later waren er ook problemen met het starten van de simulator vanuit Jenkins. Na lang zoeken werd er op internet een oplossing gevonden, als Jenkins via Homebrew (een packagemanager voor OS X) wordt geïnstalleerd zou alles moeten werken, en dat deed het ook. Veel van deze ‘kleine’ problemen zorgden ervoor dat alles net iets langer duurde dan dat er aanvankelijk voor was ingepland.

Met een aantal werkende testcases die door de tester waren ontwikkeld kon de server een tijd draaien (‘Afbeelding 9-10 Jenkins takenoverzicht’) en kijken wat er voor problemen worden gevonden.

All

| S | W | Name ↓ | Laatste geslaagd | Laatst mislukte | Duur laatste project uitvoer |
|---|---|---|---|---|---|
| | | [redacted]NDROID_GIT_PULLER | 6 uren 38 minuten - #257 | 2 dagen 6 uren - #228 | 45 seconden |
| | | [redacted]NDROID_TRIPPPLETEST | 1 maand 8 dagen - #32 | 6 uren 37 minuten - #306 | 1 minuut 1 seconde |
| | | [redacted]OS_GIT_PULLER | 26 minuten - #531 | 1 maand 13 dagen - #1 | 3.4 seconden |
| | | [redacted]OS_TRIPPPLETEST_IPHONE_7.1 | N.B. | 25 minuten - #426 | 1.8 seconden |
| | | [redacted]OS_TRIPPPLETEST_IPHONE_7.1 _SIMULATOR | 17 dagen - #172 | 25 minuten - #564 | 39 seconden |
| | | TRIPPPLETEST_GIT_PULLER | 6 dagen 11 uren - #47 | 1 maand 13 dagen - #3 | 1 minuut 2 seconden |

Afbeelding 9-10 Jenkins takenoverzicht

# 10 Testframework omzetten

*'Change is the only constant' is één van de motto's van IceMobile. Dat werd duidelijk toen er vanuit de opdrachtgevers een nieuwe opdracht kwam. Mocha en Yadda zouden moeten worden vervangen door CucumberJS. Dit zou voor vertraging zorgen op één of andere manier. En aangezien het project niet meer heel lang duurt vanaf dit punt zullen er keuzes gemaakt moeten worden in wat er wel en wat er niet geïmplementeerd gaat worden.*

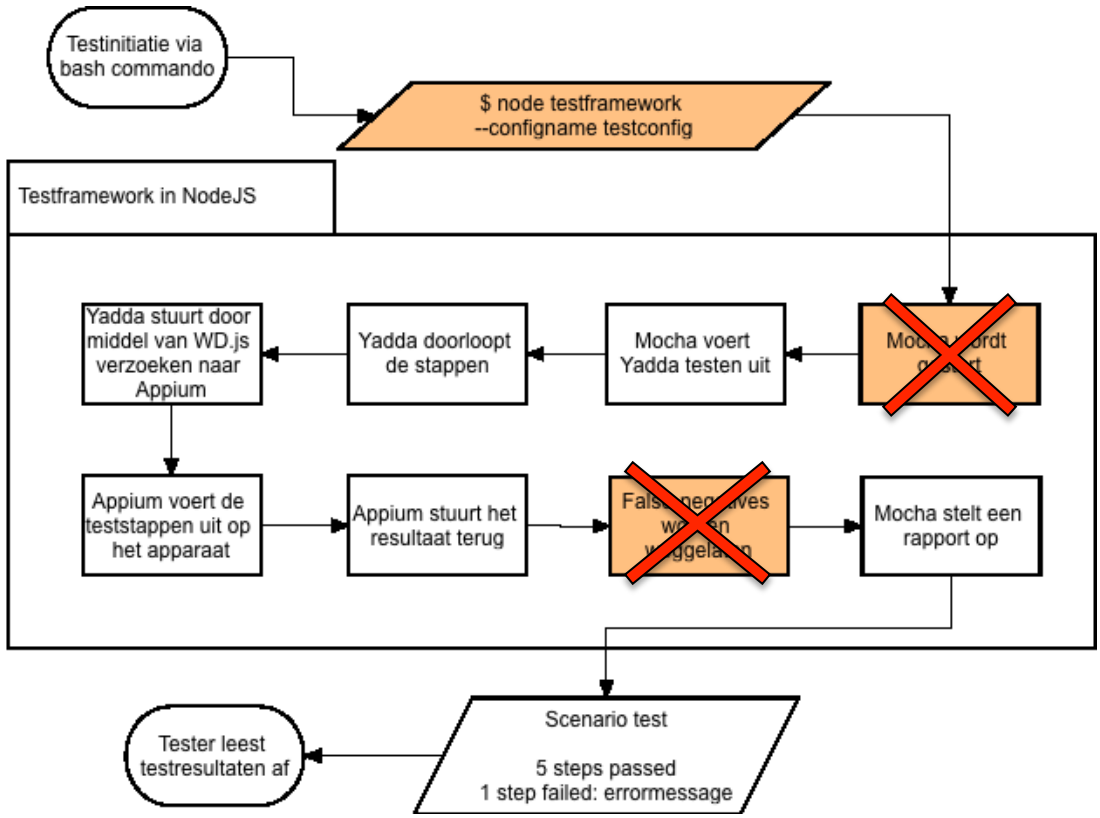
## 10.1 Onverwachte verandering

Eerder in dit verslag (paragraaf 6.2) is te lezen dat CucumberJS en Yadda beide voldoen voor dit project. Beide hebben hun, geringe, voor- en nadelen maar vanwege het feit dat de opdrachtgever stellig was in het gebruik van Yadda is er voor Yadda gekozen. Wat in principe geen probleem was totdat het hele testteam van IceMobile een overstap maakte naar CucumberJS en Yadda opeens niet meer voldeed.

Dit was een grote aanslag op de backlog van het project. Er zouden immers een groot aantal taken bij komen. Zowel voor refactoring als voor hele nieuwe code. De afstudeerder heeft aangegeven bij de opdrachtgevers dat nu waarschijnlijk niet de hele backlog leeg zou raken. Hier waren de opdrachtgevers in eerste instantie niet blij mee maar begrepen dat een switch een hoop extra werk met zich mee zou brengen. De opdrachtgevers hebben er uiteindelijk voor gekozen om op dit moment te stoppen met de verdere ontwikkeling van de CI-server en eerst het testframework klaarmaken voor CucumberJS, wetende dat de backlog misschien niet compleet wordt afgewerkt.

## 10.2 Belangrijkste keuzes vooraf

Het voordeel is dat er in het vooronderzoek al kennis is opgedaan van CucumberJS. Daarnaast is de input en output niet heel anders dan bij de combinatie Yadda en Mocha. Maar helemaal hetzelfde is het niet. Het probleem met CucumberJS is dat het niet vanuit de code kan worden aangeroepen. Alleen via een bash-script. Er moest dus een andere manier komen om het framework te starten. (‘Afbeelding 10-1 Te vervangen stappen’)




Afbeelding 10-1 Te vervangen stappen

### 10.2.1 Nieuwe taakuitvoerder

CucumberJS kan direct worden aangeroepen, dat wil zeggen: Met een simpel commando/bash-script worden de testen geïnitieerd en de rapporten achteraf opgebouwd. Hier is in principe niks mis mee behalve dat CucumberJS een aantal dingen doet die anders zouden moeten in het testframework omwille van wensen of de integratie met de CI-server.

Voor NodeJS bestaat er een taakuitvoerder die veel wordt gebruikt, namelijk Grunt. In het NodeJS maak je een Gruntfile (bijlage V) aan waar alle configuratie in staat omwille van de taken die moeten worden uitgevoerd. Daarna kan er een Grunttask worden aangemaakt die bestaat uit een aantal taken die eerder zijn geconfigureerd. Grunt voert deze taken

dan vervolgens in chronologische volgorde uit (‘Afbeelding 10-2 Grunttask’).

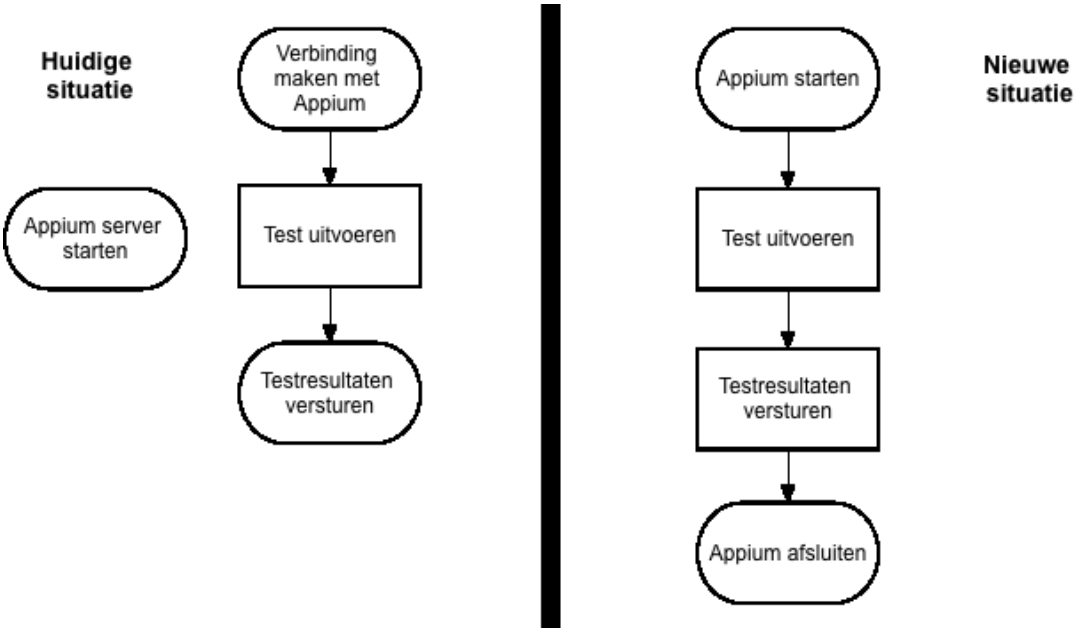
```
grunt.registerTask('test', [ 'exec:remove'
                           , 'exec:copy'
                           , 'start-appium'
                           , 'continueOn'
                           , 'cucumberjs'
                           , 'continueOff'
                           , 'json-reformatter'
                           , 'copy-test-results'
                           , 'stop-appium' ]);
```

Afbeelding 10-2 Grunttask

In het ‘Afbeelding 10-2 Grunttask’ is de taak ‘test’ te zien. Deze taak is te starten door naar de map van het NodeJS project te gaan en het commando ‘Grunt test’ uit te voeren. Vervolgens gaat er eerst iets verwijderd en gekopieerd worden, wordt Appium gestart et cetera. Door middel van Grunt kan er om CucumberJS heen extra software worden gebouwd waar nodig.

### 10.2.2 Appium server integreren

Nu er toch veel verandert moet worden omtrent het testframework is er gekozen om de Appium server niet meer los te starten voor er een test moet worden uitgevoerd maar op te nemen in de Gruntfile. Op deze manier zijn er ook geen problemen meer met sessies (paragraaf 8.5.3.2) omdat er elke keer maar één sessie wordt gestart. (‘Afbeelding 10-3 Integratie Appium server’)




Afbeelding 10-3 Integratie Appium server

## 10.3 Werkwijze

De eerste stap is het converteren van de configuratiebestanden van Mocha naar CucumberJS. Daarna de testbestanden aanpassen van Yadda en een eerste test te starten met CucumberJS. Zodra dit werkt kunnen de benodigde taken in Grunt worden geschreven en de koppeling met Jenkins weer werkend worden gemaakt. De tester gaat op dit moment op vakantie voor twee weken en ook de NodeJS experts hebben het drukker in deze periode waardoor er minder resources beschikbaar zijn. Het doel is om binnen twee weken, voordat de tester terug is, het testframework stabiel te hebben met CucumberJS en het nieuwe testframework in de CI-omgeving te hebben geplaatst.

## 10.4 Ontwikkeling

Eigenlijk het enige wat in CucumberJS zelf verandert moest worden waren de methodes die net iets anders werken dan Mocha. Daarnaast zijn de testbeschrijvingen een klein beetje anders dan die van Yadda. Dit was allemaal vrij snel voor elkaar en er waren geen tot weinig problemen.

### 10.4.1 Configuratienaam kiezen

Vanwege technische redenen kon de configuratie naam niet meer worden doorgegeven als argument, bij het starten van het testproces, maar moest dit nu als omgevingsvariabelen. Grunt geeft namelijk niet standaard de argumenten door aan de Grunttasks. Omgevingsvariabelen zijn overal uit te lezen dus moest dit veranderd worden. Het starten van een test verandert op deze manier niet heel veel (‘Afbeelding 10-4 Commando om het testframework te starten’). Net als de test zelf (bijlage X).

```
Huidige situatie:
$ node testframework –configName configurationX

Nieuwe situatie:
$ CONFIG=configurationX grunt test
```

Afbeelding 10-4 Commando om het testframework te starten

### 10.4.2 Testsuites

In de huidige situatie is er een structuur gebouwd waarbij alleen de teststappen worden ingeladen die bij de test horen die op dat moment wordt gedraaid. Zo is er voor zowel iOS en Android maar één scenariobeschrijving nodig. Afhankelijk van het platform waar de test op wordt uitgevoerd wordt dan de juiste testimplementatie geladen. Dit voorkomt dubbele scenariobeschrijvingen en zorgt er tegelijk voor dat de zelfde scenario’s op beide platformen hetzelfde moeten zijn.

CucumberJS heeft een World object. Dit is niet meer dan een global object waar alle stappen in kunnen worden gezet die voor elke test beschikbaar moeten zijn. Denk aan de testinitiëring of het afsluiten van een test. Maar ook: zoek veld x en voer tekst y in dat veld of vind knop x en klik er op.

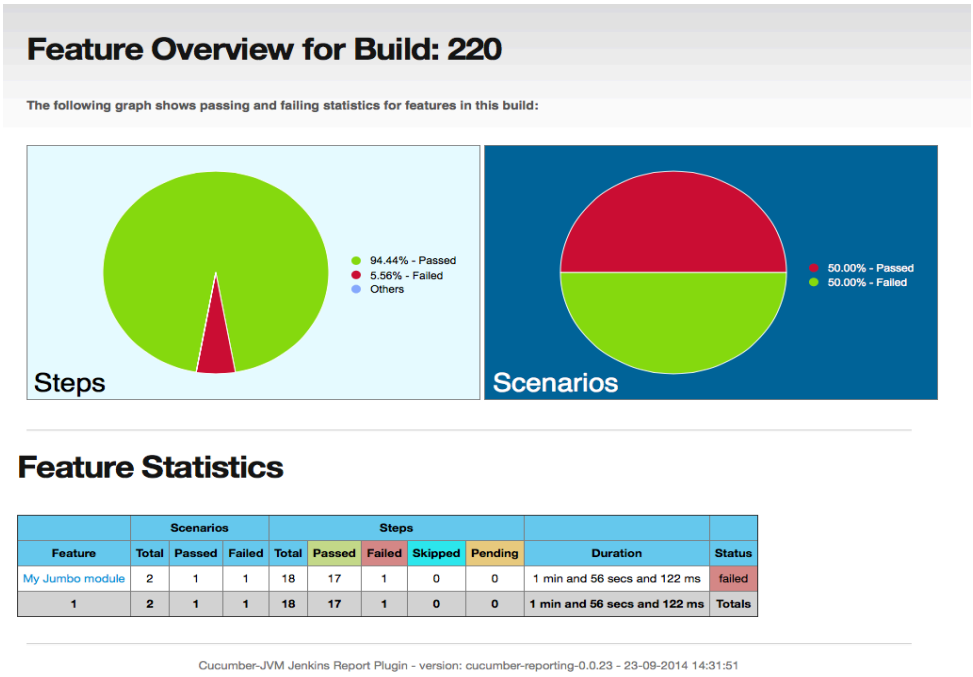
Vanwege de eigen oplossing voor het maken van testsuites wordt dit World object niet standaard ingeladen. Er is daarom een aparte Grunttask aangemaakt die er voor zorgt dat dit World object vindbaar is en wordt ingeladen.

### 10.4.3 Andere rapportbestanden

Mocha gaf aan het einde van een test een Xunit-bestand die kan worden ingeladen door Jenkins (CI-omgeving). CucumberJS werkt met een JSON bestand die standaard niet kan worden ingeladen door Jenkins. De community van zowel Jenkins als CucumberJS is groot en er werd al snel een oplossing gevonden in de vorm van een plugin voor Jenkins. Na het installeren van deze plugin kan er net als eerst worden verwezen naar het bestand met de testresultaten, alleen is dat bestand nu een JSON bestand in plaats van een Xunit bestand.

### 10.4.4 Testrapportage

Voor Jenkins is er ook een speciale plugin voor Cucumber testrapporten. Deze plugin biedt een mooie GUI (‘Afbeelding 10-5 Jenkins Cucumber plugin GUI’) aan de testers en biedt daarnaast ook een aantal extra statistieken en features.




Afbeelding 10-5 Jenkins Cucumber plugin GUI

Zo kunnen er met deze plugin screenshots worden toegevoegd aan het testrapport. Deze screenshots kunnen met CucumberJS met een speciale functie aan het rapport worden toegevoegd als base64-string (‘Afbeelding 10-6 Codevoorbeeld screenshot toevoegen’ en ‘Afbeelding 10-7 Screenshot in testrapport’). Een base64-string is niet meer dan een string van ASCII-tekens in plaats van binaire code. Bijlagen in e-mails worden ook op deze manier verstuurd en tegenwoordig biedt elke moderne webbrowser ondersteuning voor base64-strings.

```
scenario.attach(screenshotBase64String, "image/png");
```

Afbeelding 10-6 Codevoorbeeld screenshot toevoegen


Afbeelding 10-7 Screenshot in testrapport

Er was echter een probleem, en dat was dat de Javascript versie van Cucumber (CucumberJS) een aantal bugs heeft omtrent deze functie. De belangrijkste was binnen een paar dagen al opgelost na een melding op GitHub. De andere bugs konden zelf opgelost worden.

Zo is er een andere bug die de bijlagen voor een tweede keer toevoegt in het testrapport. Omdat de bug niet gevonden kon worden in CucumberJS is er gekozen om het JSON bestand achteraf uit te lezen en aan te passen ('Afbeelding 10-8 Codevoorbeeld JSON reformatter').

```
var jsonReportPath = path.resolve(process.cwd(), "test/test-results
                                                    /report.html.json");
var json = JSON.parse(fs.readFileSync(jsonReportPath, 'utf8'));

//do for every feature in the json file
_.each(json, function(feature) {
    //and every scenario within that feature file
    _.each(feature.elements, function(scenario) {
        // if the first step is the before step
        if (scenario.steps[0].keyword.toLowerCase().indexOf('before') > -1){
            if (scenario.steps[0].hasOwnProperty('embeddings')) {
                // a deletion of all the attachments fot that step
                delete scenario.steps[0].embeddings;
            }
        }
    });
});
//write the json back to where it belongs!
fs.writeFileSync(jsonReportPath, JSON.stringify(json), 'utf8');
```

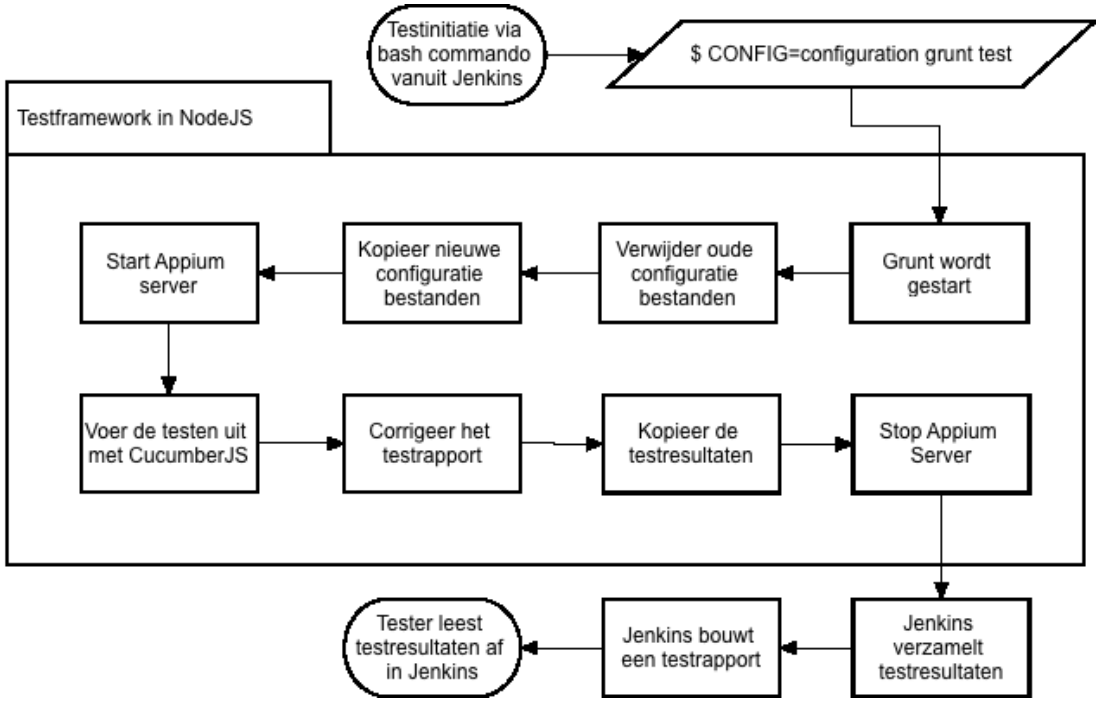
Afbeelding 10-8 Codevoorbeeld JSON reformatter

### 10.4.5 Optimalisaties

Nadat het testframework was geïnstalleerd op de CI-server zijn alle instellingen goed nagelopen voor Android en iOS. De CI-server had hier en daar nog wat tweaks nodig, zoals het werkend krijgen van fysieke apparaten voor iOS. Dit werkte nog niet omdat er abrupt moest worden begonnen met de migratie naar CucumberJS. Na een aantal optimalisaties in het testframework, verholpen bugs en optimalisaties in Jenkins kon de CI-server eindelijk stabiel draaien. En dat alles binnen de geoogde planning van twee weken.

## 10.5 Overzicht nieuwe situatie

Aangezien er een hoop is veranderd in deze periode wordt er in ‘Afbeelding 10-9 Workflow testframework nieuwe situatie’ de nieuwe situatie geschetst voor het testframework. De flow van de CI-server verandert verder niet.




Afbeelding 10-9 Workflow testframework nieuwe situatie

# 11 Jira koppeling

*Jira is een projectmanagement tool, een issuetracker. Jira wordt door IceMobile gebruikt als digitaal scrumbord+. Jira biedt veel meer functies dan alleen een statisch bord met stickies en magneetjes. Binnen Jira kunnen er automatisch burn down charts worden gegenereerd en kan een SCRUM master van achter zijn of haar computer alle lopende projecten en meldingen zien zonder dat hij of zij het hele gebouw door moet lopen. Jira biedt ook een uitgebreide API aan. De laatste stap van dit project is het vinden van een goede oplossing om bugs direct in Jira te plaatsen wanneer dit gewenst is.*

## 11.1 Keuzes en aanpak

Er is op dit moment eigenlijk maar één goede plek voor een knop om een bug te plaatsen in Jira. En dat is in het testrapport van Jenkins bij het juiste scenario. Hier bekijkt de tester namelijk de testresultaten en kan de tester direct beslissen of het gefaalde scenario daadwerkelijk een nieuwe bug is.

Zowel CucumberJS als de plugin die wordt gebruikt in Jenkins voor de testresultaten bieden hier standaard geen ondersteuning voor. Er moet dus eerst worden onderzocht wat de beste manier is om deze koppeling te maken.

## 11.2 Onderzoek

Met CucumberJS kunnen screenshots worden toegevoegd als bijlagen in het testrapport. Jenkins kan deze dan vervolgens weergeven. Al snel ontstond het idee om bij een gefaalde test, naast een screenshot, ook een stukje HTML mee te sturen met de Jira knop (‘Afbeelding 11-1 HTML als bijlage toevoegen’).

```
scenario.attach(screenshotBase64String, "image/png");
scenario.attach(htmlButtonBase64String, "text/html");
```

Afbeelding 11-1 HTML als bijlage toevoegen

Hiervoor moet wel de plugin voor Jenkins die het rapport opbouwt worden aangepast omdat deze nu puur en alleen gericht is op screenshots. Gelukkig is de plugin open source en dus aan te passen.

## 11.3 Ontwikkelen

Voor deze functionaliteit moeten er twee onderdelen worden aangepast. De plugin in Jenkins die verantwoordelijk is voor het opbouwen van het rapport en het testframework. Daarnaast is er een nieuw onderdeel nodig, de Jira connector.

### 11.3.1 Cucumber reporting plugin

Jenkins draait op Java, net als alle plug-ins van Jenkins. Alle Jenkins plug-ins zijn Maven projecten. Het eerste probleem had te maken met de Maven dependencies, die waren simpelweg niet meer vindbaar in de opgegeven repositories. Na lang zoeken is de dependency als nog gevonden en kon het project worden gebouwd.

In de html bouwer van de plugin is er een stuk verantwoordelijk voor de verwerking van de screenshots. Dit stuk is verder uitgebreid met een switch voor andere MIME-types zodat er ook HTML kan worden meegestuurd (‘Afbeelding 11-2 Codevoorbeeld toevoegen ondersteuning andere MIME-types’).

```
if (mimeHtml.equals(((LinkedTreeMap) image).get("mime_type"))){
     String mimeEncodedHtml = mimeEncodeEmbededHtml(base64String);
     links = links + "<object type=\"text/html\" id=\"htmlFrame\"
             style=\"border: none;\"  width=\"100%\" data=\"" +
             mimeEncodedHtml + "\"></object>" + "\n";
}
```

Afbeelding 11-2 Codevoorbeeld toevoegen ondersteuning andere MIME-types

Na de plugin te hebben aangepast en opnieuw te hebben gecompileerd is deze geïnstalleerd op de Jenkins server.

### 11.3.2 Testframework

In het testframework moet de HTML code worden gemaakt. Dit is gewoon een normale knop met een link eronder. De link wijst naar een webserver die in het volgende paragraaf wordt toegelicht. Met de link worden er een aantal variabelen meegestuurd:

- De naam van de feature file met de zojuist gefaalde test
- Het regelnummer van het scenario dat zojuist is gefaald
- De naam van de Jenkins taak die nu wordt uitgevoerd
- Het nummer van de Jenkins taak (run 34 bijvoorbeeld)
- De hyperlink naar de Jenkins taak

Deze set variabelen zijn de minimale vereisten om later het juiste testrapport op te halen vanuit Jenkins en een bug te melden in het juiste Jira project.

### 11.3.3 Jira connector

De Jira connector is niets meer dan een compacte webserver (NodeJS Express) die na de klik op de 'zet in Jira'-knop het gefaalde scenario in Jira zet.

Om toegang te krijgen tot de Jira API dient een tester zich eerst authentiseren. Een inlogformulier wordt getoond wanneer er op de 'zet in Jira'-knop wordt geklikt. De inloggegevens worden opgeslagen in het sessieobject van de webserver en hoeven daarom maar éénmalig te worden opgegeven.

Nadat er is ingelogd wordt het volgende request gedaan op de API van Jira ('Afbeelding 11-3 Webrequest nieuw issue in Jira').

```
var request = {
        url: config.url + 'issue/',
        method: 'POST',
        auth: {
            'user': opts.user,
            'pass': opts.pass,
            'sendImmediately': true
        },
        body: issue,
        json: true,
        followRedirect: true
};
```

Afbeelding 11-3 Webrequest nieuw issue in Jira

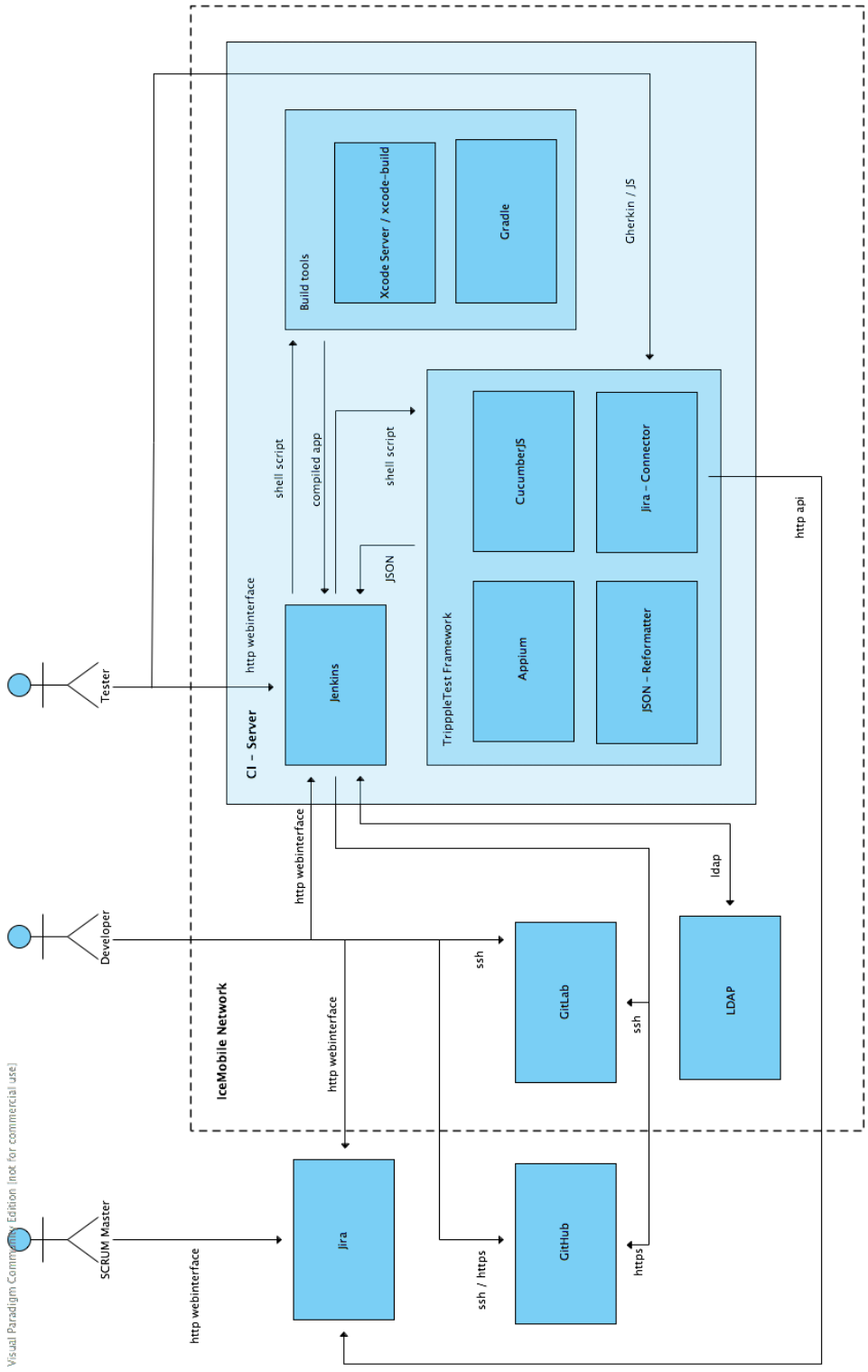
Vervolgens krijgt de tester een eventuele foutmelding te zien als de melding van de bug niet goed is gegaan. Als de melding wel goed is gegaan krijgt de tester de link naar de bug voorgeschoteld in Jira ('Afbeelding 11-4 Feedback na succesvolle melding in Jira').


Afbeelding 11-4 Feedback na succesvolle melding in Jira

In bijlage VI is te zien hoe deze melding er dan vervolgens uitziet.

# 12 Complete overzicht

*Nu er een systeem staat met vele verschillende verbindingen en frameworks is er tot slot ter verduidelijking een overzicht toegevoegd* (‘Afbeelding 12-1 Overzicht CI-omgeving IceMobile’).




Afbeelding 12-1 Overzicht CI-omgeving IceMobile

# 13 Overige bijzonderheden

*Naast de eerder beschreven werkzaamheden zijn er nog een aantal zaken die het vermelden waard zijn. Het gaat onder andere om de laatste overdracht, handleidingen schrijven, een mislukt dashboard en de CI cirkel.*

## 13.1 Laatste sprint, drukte...

Gedurende de laatste sprint, die twee weken duurde, zijn er veel dingen gebeurd.

- De Jira plugin is ontwikkeld
- Het testframework is overgedragen
- De CI-omgeving is overgedragen
- De CI-server is opnieuw geïnstalleerd voor een handleiding (bijlage VII)
- Er is een handleiding geschreven voor het testframework (bijlage VIII)
- IceMobile stapte met alle iOS projecten over op iOS 8 en Xcode 6 wat een aantal complicaties opleverde
- Er was een halve dag minder beschikbaar vanwege een tussentijds assessment
- Er was een halve dag minder beschikbaar vanwege de laatste demo aan IceMobile

Er is in de laatste dagen veel gebeurd en ook veel overgewerkt om de deadline toch te halen. Nu is dit niet de plek om te evalueren maar ik ga toch even kort wat zeggen. Ik heb hier persoonlijk onwijs veel van geleerd. Vooral op het gebied van doorzettingsvermogen. Ik raak niet snel gestrest maar begon hier de druk toch wel erg goed te voelen.

## 13.2 Testen op basis van screenshots

Naast de opgeleverde producten was er ook een wens om te kunnen testen op basis van screenshots. Als de screenshot dan bijvoorbeeld voor 2% niet overeen komt met een andere screenshot moet de test falen. Dit is vanwege tijdgebrek, door onder andere de CucumberJS migratie, niet gelukt om te implementeren.

## 13.3 Mislukt dashboard

Waar nu Jenkins wordt gebruikt als dashboard was dit niet het oorspronkelijke plan. Het idee was om een eigen dashboard te ontwikkelen met een aantal extra functies. Met deze ontwikkeling is begonnen maar

vanwege tijdgebrek en miscommunicatie tussen de opdrachtgevers en afstudeerder was dit niet wat men er van verwacht had. Met het dashboard kan er onder andere een JSON configuratie bestand van de CI-server via een website worden aangepast. Maar ook featurefiles kunnen zo worden aangepast. In bijlage IX zijn een aantal screenshots opgenomen.

## 13.4 De CI cirkel

Met de trigger bij een nieuwe commit van een developer wordt de hele CI-omgeving opgestart. Als de tester een melding krijgt van een gefaald testscenario kan hij of zij deze in Jira zetten met een druk op de knop. De tester vindt deze bug-melding in Jira als losse taak en lost de bug op. Vervolgens stuurt de developer zijn weer op naar GitLab en begint de CI cirkel weer opnieuw.

Dit is de kracht van testautomatisering en nu dus ook voor mobile bij IceMobile (zie afbeelding op de voorkant van dit verslag).

# 14 Bespreking van de opgeleverde producten

*Tijdens de stageperiode zijn er een aantal (deel)producten opgeleverd. Deze producten worden kritisch besproken in dit hoofdstuk.*

## 14.1 Testframework

Eigenlijk heb ik twee keer het testframework ontwikkeld. Het framework zag er met CucumberJS namelijk 'heel' anders uit. De werking was verreweg hetzelfde maar technisch pakten beide frameworks een aantal zaken op een andere manier aan. Daarnaast is een versie 2 altijd beter dan een versie 1. Vandaar dat ik de frameworks los van elkaar bespreek.

### 14.1.1 Versie Yadda en Mocha

Dit ontwikkeling van dit product is gestart zonder enige kennis van de materie die benodigd was voor het testframework. Het enige waar ik al eerder mee had gewerkt is NodeJS.

Ik ben tevreden over de combinatie Yadda, Mocha en Appium in het algemeen aangezien er praktisch nog geen ondersteuning en informatie beschikbaar was op het moment van starten met de ontwikkeling. Ik had het idee dat er weinig tot geen mensen deze combinatie gebruikten. Dit kwam denk ik enerzijds doordat Appium nog zéér onstabiel was en nog relatief weinig werd gebruikt (lees: weinig documentatie beschikbaar) anderzijds doordat Yadda een vrij kleine community betreft. Daarnaast liep de NodeJS client voor Appium, WD.js, achter op de andere clients. Deze combinatie maakte het lastig om een goedwerkende basis neer te zetten.

Het testframework deed wat het moest doen maar liet hier en daar nog wel een aantal steken vallen. Zo had ik gekozen voor het gebruik van een zelfgemaakte taskrunner omdat de verwachting was dat er weinig extra code toegevoegd zou moeten worden. Doordat veel van de frameworks nog verre van volwassen waren moesten er veel kleinigheden worden overwonnen. Denk aan bijvoorbeeld slecht opgebouwde testrapporten of falende testservers. Ik had hier achteraf beter een bekende taskrunner voor kunnen gebruiken, zoals bijvoorbeeld Grunt (waar ik later in het project overigens wel gebruik van maak).

Volgende keer bij het ontwikkelen van een testframework dat hier veel op lijkt ga ik zeker weer een BDD testframework gebruiken in combinatie met NodeJS. Javascript is een relatief simpele taal om voor te ontwikkelen en BDD maakt de testen een stuk duidelijker voor iedereen.

Het is moeilijk dit product te beoordelen omdat de ontwikkeling van het testframework eerder is gestopt door een switch naar CucumberJS. Ik denk dat het product in deze fase (ook Appium was nog minder ver ontwikkeld hier) nog niet productiewaardig was.

### 14.1.2 Versie CucumberJS

Deze versie is eigenlijk versie 2 van het testframework. Omdat er zoveel veranderd moest worden vanwege structuur konden een groot aantal functies opnieuw worden beschreven en oogde het geheel een stuk schoner en netter qua code. Daarnaast was mijn kennis omtrent de frameworks veel groter dan aan het begin van het project wat het makkelijker maakte om technische keuzes te maken.

Ik ben erg tevreden over de stukjes rondom CucumberJS heen. Zo kan er zonder poespas een fysiek apparaat worden gebruikt om op te testen in plaats van de apparaat-id's op te geven. Dit kan uiteraard ook nog steeds in het geval dat er meerdere apparaten aan de CI-server komen te hangen. Maar ook het deel rondom de JSON rapporten vind ik zeer geslaagd. Er kunnen door een aantal kleine aanpassingen bijlagen in verschillende formaten worden toegevoegd. Daarnaast zijn er een aantal fouten in de opbouw van de JSON rapporten gehaald waardoor ze nu correct worden verstuurd naar Jenkins.

Ook de keuze om Appium te integreren pakte goed uit, zo heb ik geen één keer meer een sessiefout van Appium (paragraaf 8.5.3.2) gevonden in deze opstelling.

Waar ik minder tevreden ben is de hechte verbinding met Jenkins. Het werkt goed maar alleen als het framework op dezelfde machine staat. Dit is misschien een mooie uitbreiding voor het framework, het testframework op afstand kunnen aansturen (zonder de CI-server uiteraard want dat werkt prima).

Volgende keer ga ik zeker weer CucumberJS gebruiken. Het werkt zeer stabiel en was een stuk makkelijk te integreren dan Yadda en Mocha. Wel zou ik de volgende keer misschien een API op het testframework bouwen zodat er op afstand zou kunnen worden gecommuniceerd met het testframework.

De verwachtingen zijn wat mij betreft zeker gehaald, ik had aan het begin niet verwacht dat ik dit zou kunnen neerzetten. Alles draait stabiel en is niet al te ingewikkeld opgebouwd.

## 14.2 CI-server

De CI-server vergt veel tijd om voor de eerste keer goed op te bouwen vanwege de vele afhankelijkheden. Een kleine nietszeggende update kan van grote invloed zijn op de rest van het systeem, denk aan het butterfly effect. De eerste test CI-server draait nog steeds stabiel en doet wat hij moet doen. Onder grote stress heeft de server nog niet gestaan vanwege het gebrek aan testen dus hoe dit uitpakt is nog niet geheel bekend maar ik voorzie hier geen grote problemen die verdere ontwikkeling in de weg zouden staan.

Het meest tevreden ben ik met de aanpassing van de Cucumber testrapportage plugin van Jenkins. Er kunnen nu een hoop andere bijlagen worden toegevoegd. Niet alleen screenshots maar ook HTML, javascript of gewoon tekst. Hier heeft niet de meeste tijd ingezeten maar de oplossing heeft zeer goed uitgepakt al zeg ik het zelf.

Volgende keer zou ik weer voor Jenkins kiezen maar dan misschien alles op een Linux machine zetten. Er waren toch een aantal bugs in Jenkins die vooral met Mac OS X te maken hadden. Als Jenkins op Linux gaat draaien moet er wel een oplossing worden gevonden om het testframework op een andere machine te zetten aangezien Xcode benodigd is voor testen op iOS en Xcode werkt alleen op Mac OS X.

Ik denk dat dit een geslaagd product is geworden. Het product is echter nog verre aan het einde van zijn ontwikkeling. Door alle afhankelijkheden van de CI-server is het daarnaast ook verstandig om het geheel in een OTAP straat te zetten.

## 14.3 Jira koppeling

Deze koppeling is eigenlijk zeer compact in gebruik en ik zou hier ook niet teveel aan willen veranderen. Het werkt prima en is goed aanpasbaar om eventueel bugs te verhelpen als die aanwezig zijn. De koppeling is erg basic en kan misschien nog wat worden uitgebreid met extra functionaliteit.

Volgende keer zou ik zeker weer een zelfde soort koppeling met een projectmanagement tool plaatsen in een CI-omgeving. Het scheelt veel tijd voor de testers.

## 14.4 Dashboard

Het dashboard, waarvan de ontwikkeling is gestopt, was misschien te gehaast in elkaar gezet. Er waren immers ook maar een paar dagen beschikbaar vanwege de onverwachte overstap naar CucumberJS. Buiten dat was het idee op zich goed maar de uitwerking matig.

Daarnaast waren de verwachtingen van de opdrachtgevers qua soort functionaliteit anders dan die van mij. De miscommunicatie is geen excuus voor het matige dashboard. Web development is sowieso iets wat mij minder ligt. Ik heb er meer tijd voor nodig dan met bijvoorbeeld middleware. En die tijd had ik nu gewoon niet vanwege verschillende gebeurtenissen binnen het project.

Volgende keer zou ik hier meer tijd voor incalculeren. Dat geldt eigenlijk voor alle front-end web applicaties. Ik zou wel dezelfde tools willen gebruiken, die werken goed en begin ik langzaam bekend mee te raken.

# 15 Evaluatie

*Het is altijd goed om stil te staan bij keuzes die gemaakt zijn gedurende een project. Terugkijken op het proces kan er voor zorgen dat er de volgende keer niet dezelfde fouten worden gemaakt maar ook kunnen de sterke momenten kunnen worden herkend. Naast de procesevaluatie beschrijf ik de raakvlakken van dit project in combinatie met de beroepstaken van mijn opleiding.*

## 15.1 Proces

Het proces is verdeeld in start, midden en afronding.

### 15.1.1 Start

Er kon snel begonnen worden met het verzamelen van de verschillende wensen van de verschillende stakeholders. Waarvan ik aan het begin nog niet precies wist wie dat zouden zijn. Is een app developer bijvoorbeeld een stakeholder? Ik heb er voor gekozen om elke functiegroep die misschien iets met het project te maken zou krijgen te bevragen. Een aantal hadden al eerder gebruik gemaakt van front end testautomatisering voor mobile apps.

De testers (ook opdrachtgevers) duwden mij in een bepaalde hoek waar ik moest beginnen met zoeken omdat zij dachten dat dat het beste zou zijn. Al snel kwamen we er, los van elkaar, achter dat er in die hoek niet te vinden was wat wij zochten. Verder onderzoek en kleine POC's zorgden voor een beter algemeen beeld van wat de opdrachtgevers wilden en wat er al beschikbaar was op het web.

In eerste instantie was het idee om álles zelf te gaan ontwikkelen en geen, of bijna geen, andere frameworks te gebruiken. Daar werd al snel van afgestapt omdat dit 1: teveel werk zou zijn voor een stage van 19 weken en 2: waarom het wiel opnieuw uitvinden als er GOEDE alternatieven aanwezig zijn. Wat verstaan we onder goed? Dat is iets waar ik mee aan de slag ben gegaan. Uiteindelijk heeft de opdrachtgever de keuze gemaakt.

### 15.1.2 Midden

Vervolgens werd er begonnen met het ontwikkeltraject. Misschien ben ik te vroeg begonnen met scrummen en was de onderzoeksfase nog niet helemaal afgerond. Er kon nog geen goede inschatting worden gemaakt van wat er precies in de backlog moest komen. Nadat de stabiliteitsproblemen waren opgelost is er eigenlijk pas echt begonnen met scrummen. En al snel kwamen er een projectmanager en tester bij wat er in resulteerde dat het project een stuk soepeler ging lopen. Eigenlijk is er

vanaf dit moment niet veel bijzonders gebeurd wat betreft het proces. Alles verliep volgens plan totdat de opdracht kwam om een overstap te maken naar CucumberJS.

Waarschijnlijk zou niet alles afkomen. Ik moest aangeven wat er haalbaar was en wat niet. Ik heb toen gezegd dat het testen aan de hand van screenshotvergelijkingen dan waarschijnlijk in de backlog zou blijven staan. Dit was oké en er is toen besloten dat de rest van de backlog wel zou af komen.

### 15.1.3 Afronding

Gedurende de laatste sprint was er nog onwijs veel werk en heb ik bijna elke dag tot 21.00 uur overgewerkt om er voor te zorgen dat alles af zou komen. Er waren in deze weken ook een aantal demo's die net iets langer duurden dan normaal vanwege de grote van de groep en de fase van het product. Daarnaast gingen er een aantal testers van verschillende projecten gedurende de laatste dagen al aan de slag met het framework wat soms een aantal extra vragen opleverde maar over het algemeen kon men zich alleen prima redden. Ik heb een aantal handleidingen geschreven die in bijlagen VII en VIII zijn terug te vinden en heb de tijd genomen om een aantal vragen te beantwoorden wat betreft 'hoe nu verder'. Na mijn stage ben ik nog een dag teruggekomen om een aantal mensen die op vakantie waren nog een kleine demo te geven. De afronding was druk maar verliep voor de rest denk ik goed.

## 15.2 Beroepstaken

Tijdens dit project heb ik met een aantal beroepstaken te maken gehad. Vooraf zijn er een aantal beroepstaken benoemd die aan bod zullen komen tijdens dit project. Echter heeft u misschien gemerkt dat er een aantal beroepstaken iets minder aan bod zijn gekomen dan aanvankelijk werd gedacht. Daarentegen zijn er een aantal andere beroepstaken juist wel aan bod gekomen. De eerste vier zijn de beroepstaken die vooraf al zijn benoemd in het afstudeerplan, daarna staat er nog een vijfde beschreven die niet vooraf in het afstudeerplan is vastgelegd maar wel in het project aan bod is gekomen.

### 15.2.1 Voorbereiden en opstarten software ontwikkeltraject (1.2)

Het project was nieuw voor mij en ook nieuw voor IceMobile. Er zou dus een apart ontwikkeltraject komen. Vooraf is er geprobeerd om goed beeld te krijgen van waar de opdrachtgevers behoefte aan hebben. Ik ben in deze fase veel bezig geweest met onderzoek, overleg met de opdrachtgevers en met werknemers die al eerder aan de slag zijn gegaan met geautomatiseerde acceptatietesten voor mobiel. Vervolgens heb ik een

scrumbord gemaakt met een backlog van de grote features/epic's die uiteindelijk in het product moeten komen. Per sprint is er een demo gegeven (een enkele keer niet, dit was vooral aan het begin). Een stand-up vond niet elke dag plaatst vanwege het feit dat ik de eerste sprint á 2 sprints alleen werkte aan dit project. Vervolgens, nog tijdens de eerste fase, kreeg ik een projectmanager en een tester die full time met het product aan de slag ging. We hebben toen elke dag een stand-up gehouden en per user story per sprint zoveel mogelijk taken bedacht.

### 15.2.2 Uitvoeren analyse door definitie van requirements (1.4)

Deze beroepstaak vind ik zelf lastig te plaatsen in een Agile Scrum project. Uiteraard zijn er vooraf een aantal wensen en eisen vastgesteld maar deze ontstonden eerder gaandeweg het project dan dat er aan het begin gedurende een ononderbroken periode een analyse is uitgevoerd. De requirements ontstonden ook door middel van de demo's die werden gegeven. Door vragen van de experts en opdrachtgevers aan mij en andersom konden er nieuwe requirements worden ontdekt. Zo kon er ook snel worden bijgestuurd als dit nodig was, bijvoorbeeld bij het ontwikkelen van het dashboard (paragraaf 13.3) waar uiteindelijk de stekker uit is getrokken.

Er is daarnaast bijna dagelijks een update gegeven aan de opdrachtgevers zodat zij wisten waar ik mee bezig was en wat zij konden verwachten.

### 15.2.3 Ontwerpen systeemdeel (3.2)

Naast een heel systeemdeel te hebben ontworpen (het testframework, hoofdstuk 8 en 10) heb ik ook het gehele CI-systeem ontworpen. Door middel van overleg met experts, het bekijken van de werking van verschillende frameworks en het maken van een aantal diagrammen heb ik tijdens het ontwikkelen zorg gedragen voor het ontwerp.

### 15.2.4 Bouwen applicatie (3.3)

Aan het begin was de veronderstelling dat álles vanaf de grond af aan zou worden opgebouwd. Al snel had iedereen door dat dit zonde van de tijd is vanwege betere alternatieven en repetitief werk. Daarnaast had ik nooit kunnen opleveren wat ik nu heb opgeleverd. Ik had misschien een basis testframework kunnen neerzetten maar had het nooit kunnen plaatsen in een CI-omgeving, daar was gewoon geen tijd voor.

Nu ben ik vooral bezig geweest met het bouwen van de software wat van de losse frameworks een framework maakt. Ze noemen dit in de NodeJS community ook wel hacking. Dit betekent overigens niet dat er meer is

geconfigureerd dan geprogrammeerd. Het blijft programmeren. In de bijlagen zijn een aantal voorbeelden van code te zien.

### 15.2.5 Ontwerpen softwarearchitectuur (3.1)

Dit is één van de beroepstaken waar ik veel meer mee bezig ben geweest dan aanvankelijk gedacht. De tijd die ik hier in heb gestopt is voornamelijk in plaats van de tijd die ik anders in beroepstaak 3.3 'Bouwen applicatie' zou hebben gestoken. Ik ben vooral bezig geweest met het feit hoe gegevens het best verplaatst konden worden van het ene systeem(deel) naar het andere. Bijvoorbeeld: de Jira koppeling. De manier om HTML verpakt als base64 string in een JSON rapport te plaatsen is misschien niet de eerste keuze waar je aan zou denken maar door deze oplossing zijn er maar minimale aanpassingen nodig in andere stukken software. Daarnaast blijven de verantwoordelijkheden van de verschillende systeemdelen hetzelfde.

Met al de gebruikte frameworks moest er een manier verzonnen worden om het testframework netjes en begrijpelijk te houden. Dit was één van de uitdagingen. De vraag was dan ook vaak 'hoe?'. Hoe plaats ik dit framework in het project, hoe haal ik de fouten uit een rapport, hoe om te gaan met een falend platform et cetera.

# Literatuurlijst

W. Bervoets & H. Mastenbroek (2009), Reader procesverslag schrijven

N. Rozanski & E. Woods (2011), Software Systems Architecture

Jira API (2014), https://docs.atlassian.com/jira/REST/latest/

Appium (2014), http://appium.io

Calabash (2014), http://calaba.sh

IphoneDriver (2014), https://code.google.com/p/selenium/wiki/IPhoneDriver

Yadda (2014), https://github.com/acuminous/yadda

Mocha (2014), http://visionmedia.github.io/mocha/

CucumberJS (2014), https://github.com/cucumber/cucumber-js

Cucumber (2014), http://cukes.info

ExpressJS (2014), http://expressjs.com

WD.js (2014), https://github.com/admc/wd

GruntJS (2014), http://gruntjs.com

Cucumber Reporting (2014), https://github.com/masterthought/cucumber-reporting

Cucumber Reporting Jenkins plugin (2014), https://github.com/masterthought/jenkins-cucumber-jvm-reports-plugin-java

Gradle (2014), http://www.gradle.org

Apache Maven (2014), http://maven.apache.org

Apple (2014), https://developer.apple.com/library/ios/navigation/

Jade (2014), http://jade-lang.com/reference

Joyent (2013-2014), http://nodejs.org/api/

Wikipedia (2014), http://en.wikipedia.org/wiki/Scrum_(software_development)

Watirmelon (2014),
http://watirmelon.com/category/automated-acceptance-testing-2/

M. Moser (2012),
http://stackoverflow.com/questions/4973981/how-to-choose-between-hudson-and-jenkins

Jenkins (2014),
https://wiki.jenkins-ci.org/pages/viewpage.action?pageId=53608972

Jenkins (2014),
https://wiki.jenkins-ci.org/display/JENKINS/Plugins

Toughts (2014),
http://thoughts.wallproductions.com/2014/01/jenkins-vs-travis/

S. Ritchie (2012),
http://www.excella.com/blog/teamcity-vs-jenkins-better-continuous-integration-server/

Utsav (2013),
http://xebee.xebia.in/index.php/2013/10/29/mobile_automation_tools/

Y. Shah (2014),
http://www.moolya.com/blogs/2014/03/36/Blended-Automation--Part-1--App-vs-Tool-Assessment-

Acuminous (2013),
http://blog.acuminous.co.uk/2013/05/do-you-like-your-bdd-full-fat-skimmed.html

O. Reminnyi & D. Krauss (2013)
http://www.devx.com/enterprise/creating-an-effective-gui-automation-framework.html

I. Dorovskikh & K. Gawande (2013)
https://docs.google.com/presentation/d/1m-rYQjOl7s9sC0HQQ5yZOyPOlrNnRZYxCEC-8UHbeLw/pub?start=false&loop=false&delayms=3000&slide=id.gbe85d8b9_0145

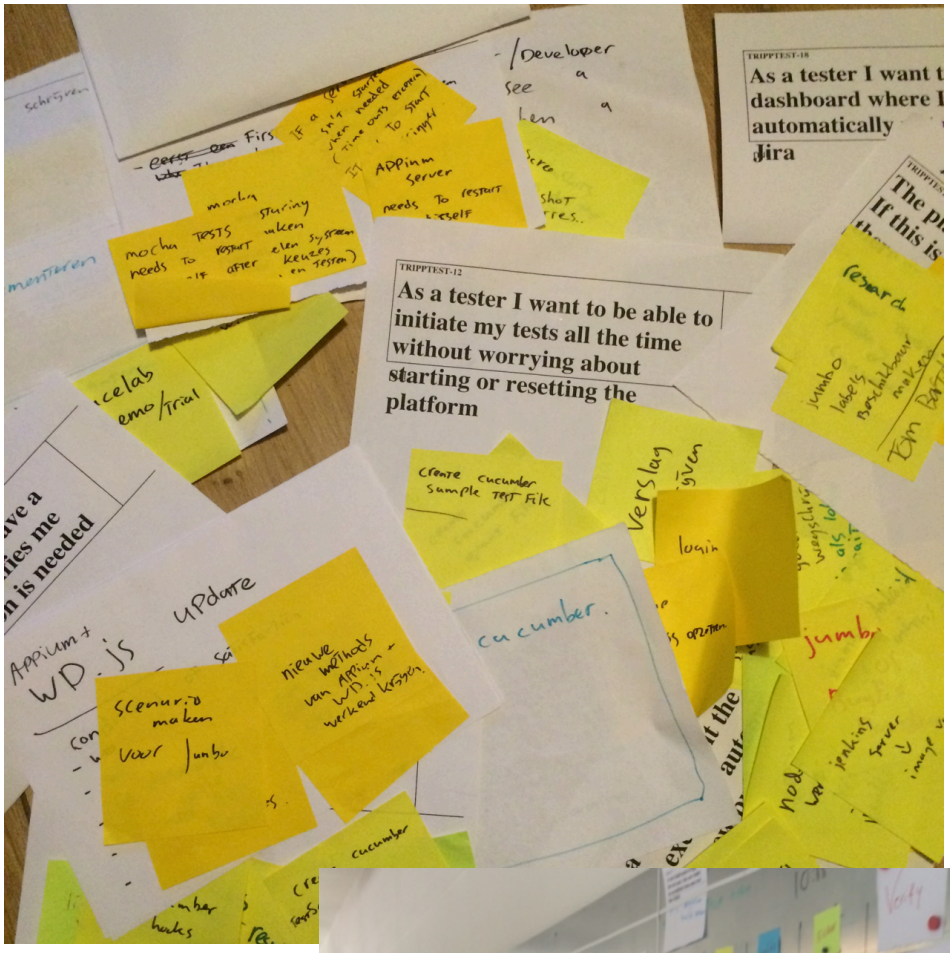
# Afbeeldingen- en codevoorbeeldenlijst

# Bijlagenlijst

*Op de volgende pagina's zijn de bijlagen te vinden.*

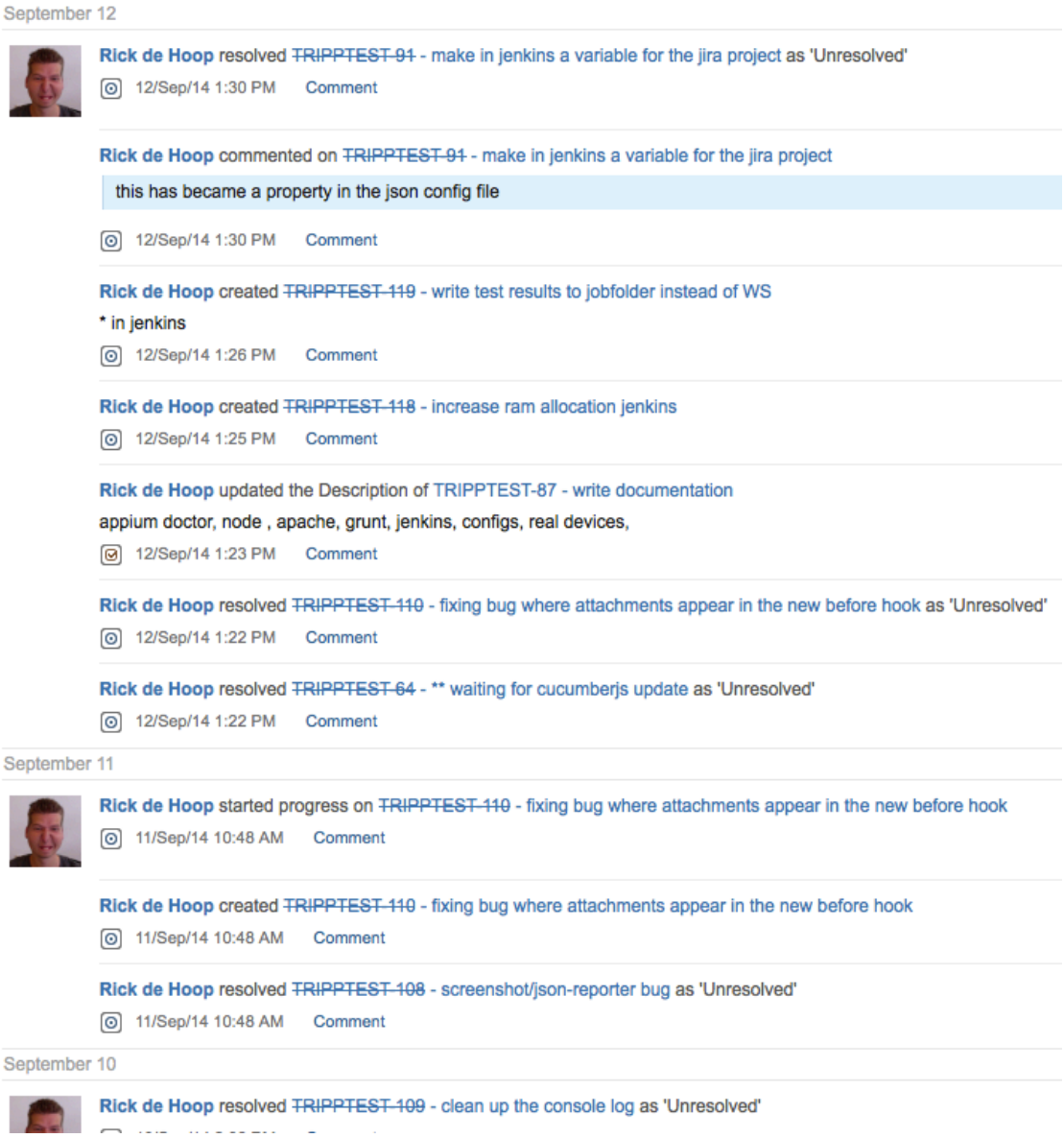
# Bijlage I Taken en user stories fysiek scrumbord

*Een tweetal foto's van het oude fysieke scrumbord.*




# Bijlage II Taken en user stories digitaal scrumbord

*Een aantal screenshots van de digitale scrum-omgeving.*



September 12

**Rick de Hoop** resolved ~~TRIPPTEST-91~~ - make in jenkins a variable for the jira project as 'Unresolved'
12/Sep/14 1:30 PM Comment

**Rick de Hoop** commented on ~~TRIPPTEST-91~~ - make in jenkins a variable for the jira project
this has became a property in the json config file
12/Sep/14 1:30 PM Comment

**Rick de Hoop** created ~~TRIPPTEST-119~~ - write test results to jobfolder instead of WS
* in jenkins
12/Sep/14 1:26 PM Comment

**Rick de Hoop** created ~~TRIPPTEST-118~~ - increase ram allocation jenkins
12/Sep/14 1:25 PM Comment

**Rick de Hoop** updated the Description of TRIPPTEST-87 - write documentation
appium doctor, node , apache, grunt, jenkins, configs, real devices,
12/Sep/14 1:23 PM Comment

**Rick de Hoop** resolved ~~TRIPPTEST-110~~ - fixing bug where attachments appear in the new before hook as 'Unresolved'
12/Sep/14 1:22 PM Comment

**Rick de Hoop** resolved ~~TRIPPTEST-64~~ - ** waiting for cucumberjs update as 'Unresolved'
12/Sep/14 1:22 PM Comment

September 11

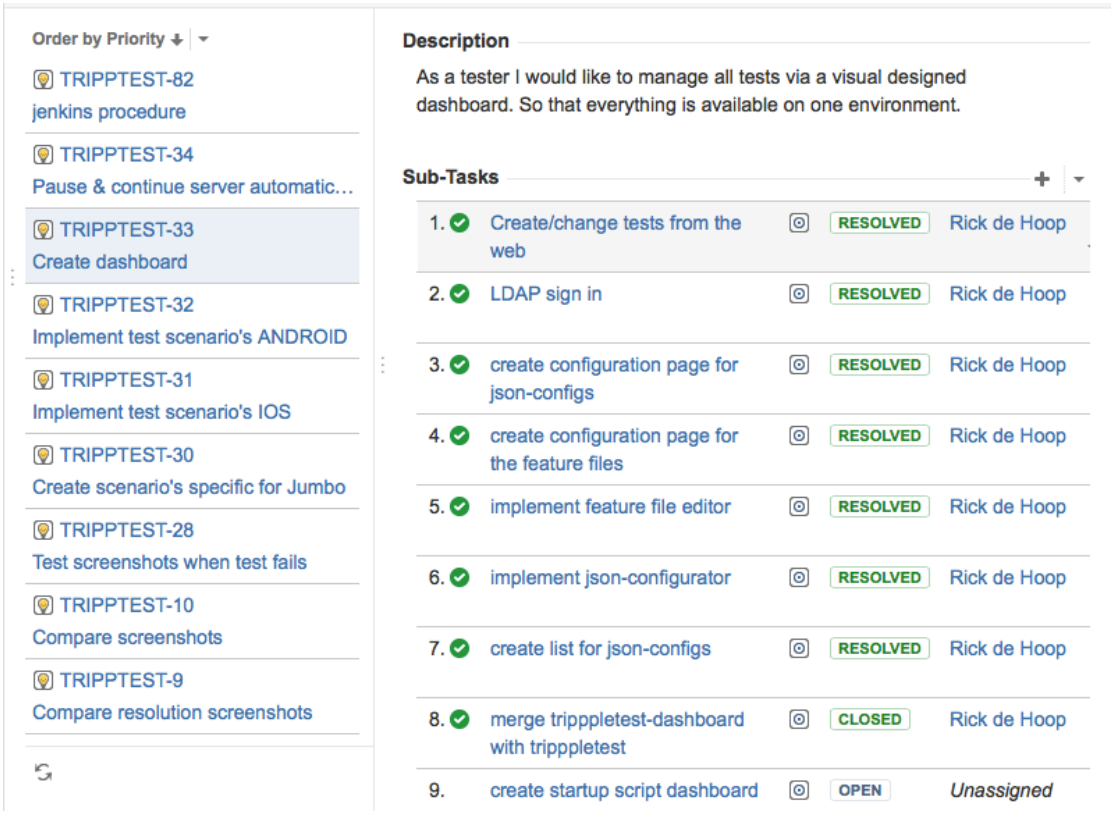
**Rick de Hoop** started progress on ~~TRIPPTEST-110~~ - fixing bug where attachments appear in the new before hook
11/Sep/14 10:48 AM Comment

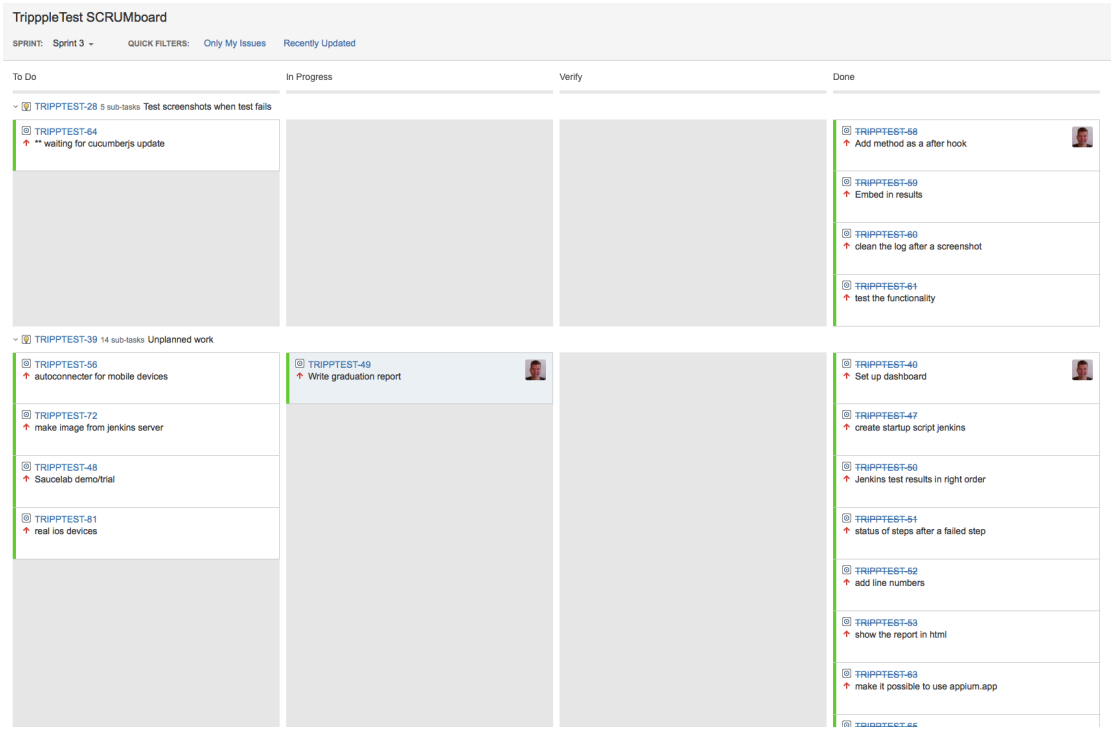
**Rick de Hoop** created ~~TRIPPTEST-110~~ - fixing bug where attachments appear in the new before hook
11/Sep/14 10:48 AM Comment

**Rick de Hoop** resolved ~~TRIPPTEST-108~~ - screenshot/json-reporter bug as 'Unresolved'
11/Sep/14 10:48 AM Comment

September 10

**Rick de Hoop** resolved ~~TRIPPTEST-109~~ - clean up the console log as 'Unresolved'


Order by Priority
TRIPPTEST-82
jenkins procedure
TRIPPTEST-34
Pause & continue server automatic...
TRIPPTEST-33
Create dashboard
TRIPPTEST-32
Implement test scenario's ANDROID
TRIPPTEST-31
Implement test scenario's IOS
TRIPPTEST-30
Create scenario's specific for Jumbo
TRIPPTEST-28
Test screenshots when test fails
TRIPPTEST-10
Compare screenshots
TRIPPTEST-9
Compare resolution screenshots
Description
As a tester I would like to manage all tests via a visual designed dashboard. So that everything is available on one environment.
Sub-Tasks
1. Create/change tests from the web RESOLVED Rick de Hoop
2. LDAP sign in RESOLVED Rick de Hoop
3. create configuration page for json-configs RESOLVED Rick de Hoop
4. create configuration page for the feature files RESOLVED Rick de Hoop
5. implement feature file editor RESOLVED Rick de Hoop
6. implement json-configurator RESOLVED Rick de Hoop
7. create list for json-configs RESOLVED Rick de Hoop
8. merge trippletest-dashboard with trippletest CLOSED Rick de Hoop
9. create startup script dashboard OPEN Unassigned


TripppleTest SCRUMboard
SPRINT: Sprint 3
QUICK FILTERS: Only My Issues Recently Updated
To Do
In Progress
Verify
Done
TRIPPTEST-28 5 sub-tasks Test screenshots when test fails
TRIPPTEST-64 ** waiting for cucumberjs update
TRIPPTEST-58 Add method as a after hook
TRIPPTEST-59 Embed in results
TRIPPTEST-60 clean the log after a screenshot
TRIPPTEST-61 test the functionality
TRIPPTEST-39 14 sub-tasks Unplanned work
TRIPPTEST-56 autoconnecter for mobile devices
TRIPPTEST-49 Write graduation report
TRIPPTEST-40 Set up dashboard
TRIPPTEST-72 make image from jenkins server
TRIPPTEST-47 create startup script jenkins
TRIPPTEST-48 Saucelab demo/trial
TRIPPTEST-50 Jenkins test results in right order
TRIPPTEST-81 real ios devices
TRIPPTEST-51 status of steps after a failed step
TRIPPTEST-52 add line numbers
TRIPPTEST-53 show the report in html
TRIPPTEST-63 make it possible to use applium.app

# Bijlage III Epic's

*De initiële lijst met epic's als uitgangspunt voor het project (Engels).*

1. **Stable test platform**
   - As a tester I want to be able to initiate my tests all the time without worrying about starting or resetting the platform.
   - As a tester I do not want 'bad/wrong' test results because of a failing test platform.
   - As a tester I want the platform to be resetted automatically in case of hick-ups or crashes.
2. **Integration with build server**
   - As a tester I want the test cases to be executed automatically when a new build is built.
3. **Add extra feature-file functionalities**
   - As a tester/developer I want to be able to re-use/extend test cases to avoid duplicate code (for example, scenarion sign in, and scenario redeem stamps. Redeem stamps has as precondition: user needs to be signed in).
   - As a tester I want to be able to have the step files and feature files combined in 1 file because off the maintainability.
   - As a tester/developer I want to create/change the test cases from a webinterface.
4. **Dashboard**
   - As a tester i want to have a dashboard where I can start my test cases.
   - As a tester i want to have a dashboard where I can see the test results (and stats).
   - As a tester i want to have a dashboard that notifies me when my attention is needed.
   - As a tester i want to have a dashboard where I can automatically put new issues in Jira.
   - As a tester i want to have a dashboard that notifies me when my attention is needed.
5. **Screenshot comparison in tests**
   - As a tester I want the ability to take screenshots from within the test cases.
   - As a tester I want to be able to compare screenshots taken during the test cases with the expected ones.
   - As a tester I want to see how and where the screenshots differ from each other.
   - As a tester/developer I want to upload a screenshot how the screen should look like.

# Bijlage IV Yadda, Mocha en Appium in actie

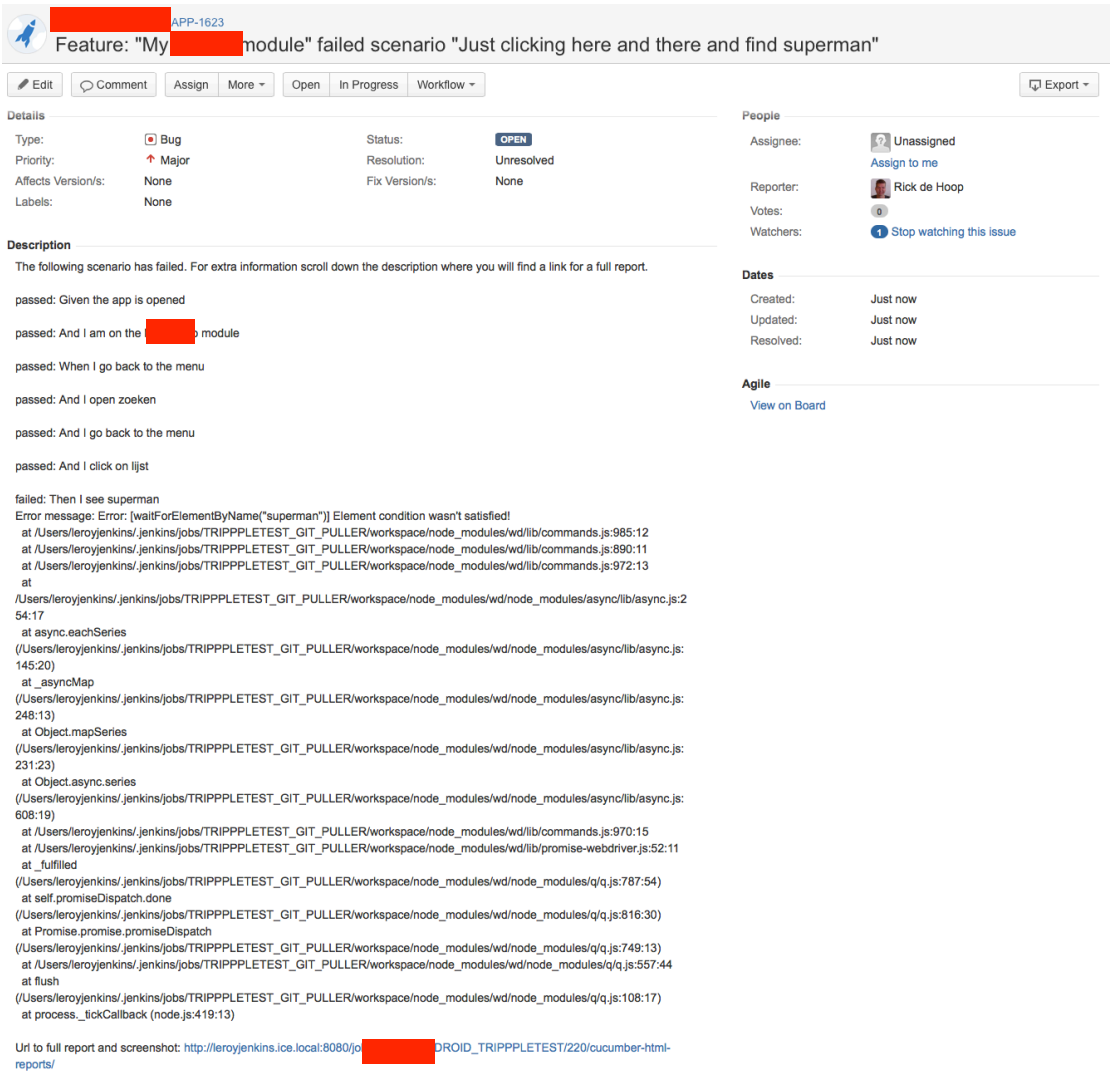
# Bijlage V Gruntfile testframework

```
module.exports = function(grunt) {

    // load modules
    var path = require('path');

    // set grunt
    grunt.initConfig({
        configJSON: grunt.file.readJSON(path.resolve(process.cwd(),
                    'config/' + getConfigLocation() + '.json')),
        exec: {
            copy: {
                cmd: 'cp -R ./test/support ./test/steps/<%= configJSON.
                      platformName %>/<%= configJSON.testSuite %>/;'
            },
            remove: {
                cmd: 'rm -rf ./test/steps/<%= configJSON.platformName
                      %>/<%= configJSON.testSuite %>/support'
            }
        },
        cucumberjs: {
            options: {
                // FORMAT: ['pretty', 'progress', 'summary', 'html']
                format: 'html',
                output: path.resolve(process.cwd(), 'test/test-results
                                                     /report.html'),
                // THEME: ['foundation', 'bootstrap', 'simple']
                theme: 'bootstrap',
                saveJson: true,
                tags: '~@todo',
                steps: './test/steps/<%= configJSON.platformName %>/<%=
                        configJSON.testSuite %>/'
            },
            features: ['test/features/<%= configJSON.testSuite %>/']
        },
        nodemon: {
            dashboard: {
                script: path.resolve(process.cwd(), 'lib/trippletest-
                                                    dashboard/app.js')
            },
            jiraconnector: {
                script: path.resolve(process.cwd(), 'lib/jira-connector
                                                    /app.js')
            }
        }
    });
```

Codevoorbeeld gaat verder op de volgende pagina...

```
    grunt.loadTasks('lib/grunt-tasks');
    grunt.loadNpmTasks('grunt-continue');
    grunt.loadNpmTasks('grunt-exec');
    grunt.loadNpmTasks('grunt-cucumberjs');
    grunt.loadNpmTasks('grunt-nodemon');

    grunt.registerTask('test', [ 'exec:remove'
                               , 'exec:copy'
                               , 'start-appium'
                               , 'continueOn'
                               , 'cucumberjs'
                               , 'continueOff'
                               , 'json-reformatter'
                               , 'copy-test-results'
                               , 'stop-appium' ]);

    grunt.registerTask('test-debug', [ 'exec:remove'
                                     , 'exec:copy'
                                     , 'start-appium'
                                     , 'continueOn'
                                     , 'cucumberjs'
                                     , 'continueOff'
                                     , 'json-reformatter'
                                     , 'copy-test-results'
                                     , 'stop-appium' ]);

    grunt.registerTask('test-appium-local', [ 'exec:remove'
                                            , 'exec:copy'
                                            , 'continueOn'
                                            , 'cucumberjs'
                                            , 'continueOff'
                                            , 'json-reformatter' ]);

    grunt.registerTask('dashboard', ['nodemon:dashboard'] );

    grunt.registerTask('jira-connector', ['nodemon:jiraconnector'] );

};

function getConfigLocation(){
    if(process.env.JENKINS){
        return 'jenkins/'+ (process.env.CONFIG || 'default_config');
    } else {
        return process.env.CONFIG || 'default_config';
    }
}

// thats all folks
```

# Bijlage VI Jira bug



[redacted]-APP-1623

Feature: "My [redacted] module" failed scenario "Just clicking here and there and find superman"

Edit | Comment | Assign | More | Open | In Progress | Workflow | Export

**Details**

| | | | |
|---|---|---|---|
| Type: | Bug | Status: | OPEN |
| Priority: | Major | Resolution: | Unresolved |
| Affects Version/s: | None | Fix Version/s: | None |
| Labels: | None | | |

**People**

Assignee: Unassigned
Assign to me
Reporter: Rick de Hoop
Votes: 0
Watchers: 1 Stop watching this issue

**Dates**

Created: Just now
Updated: Just now
Resolved: Just now

**Agile**

View on Board

**Description**

The following scenario has failed. For extra information scroll down the description where you will find a link for a full report.

passed: Given the app is opened

passed: And I am on the [redacted] module

passed: When I go back to the menu

passed: And I open zoeken

passed: And I go back to the menu

passed: And I click on lijst

failed: Then I see superman
Error message: Error: [waitForElementByName("superman")] Element condition wasn't satisfied!
at /Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/lib/commands.js:985:12
at /Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/lib/commands.js:890:11
at /Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/lib/commands.js:972:13
at /Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/async/lib/async.js:254:17
at async.eachSeries (/Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/async/lib/async.js:145:20)
at _asyncMap (/Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/async/lib/async.js:248:13)
at Object.mapSeries (/Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/async/lib/async.js:231:23)
at Object.async.series (/Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/async/lib/async.js:608:19)
at /Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/lib/commands.js:970:15
at /Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/lib/promise-webdriver.js:52:11
at _fulfilled (/Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/q/q.js:787:54)
at self.promiseDispatch.done (/Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/q/q.js:816:30)
at Promise.promise.promiseDispatch (/Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/q/q.js:749:13)
at /Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/q/q.js:557:44
at flush (/Users/leroyjenkins/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/node_modules/wd/node_modules/q/q.js:108:17)
at process._tickCallback (node.js:419:13)

Url to full report and screenshot: http://leroyjenkins.ice.local:8080/jo[redacted]DROID_TRIPPPLETEST/220/cucumber-html-reports/

# Bijlage VII Handleiding installatie CI-server

ci-server-installation/ at master - Ben de Laat / ci-server installation | GitLab 04-10-14 21:01

| Name | Last Update | Last Commit › 65f71d51a73 (/ben/ci-server-installation/commit/65f71d51a730bd720dd589d23d2c725432c9348c) – worked on readme | history (/ben/ci-server-installation/commits/ma |
|---|---|---|---|
| cucumber-reporter-source (/ben/ci-serv… | 15 days ago | Rick de Hoop (/u/rick) added source for jenkins plugin without the git repo (/ben/ci-ser… | |
| files (/ben/ci-server-installation/tree/mas… | 15 days ago | Rick de Hoop (/u/rick) added extra ram to the jenkins job in the config file (/ben/ci-serv… | |
| jenkins-jobs (/ben/ci-server-installation/t… | 15 days ago | Rick de Hoop (/u/rick) added config files for jobs (/ben/ci-server-installation/commit/c9… | |
| jenkins-scripts (/ben/ci-server-installatio… | 15 days ago | Rick de Hoop (/u/rick) added the xml with mamanaged scripts (/ben/ci-server-installati… | |
| .DS_Store (/ben/ci-server-installation/blo… | 15 days ago | Rick de Hoop (/u/rick) added extra ram to the jenkins job in the config file (/ben/ci-serv… | |
| .gitignore (/ben/ci-server-installation/blo… | 15 days ago | Rick de Hoop (/u/rick) working on readme (/ben/ci-server-installation/commit/c85a34a… | |
| readme.md (/ben/ci-server-installation/bl… | 11 days ago | Rick de Hoop (/u/rick) worked on readme (/ben/ci-server-installation/commit/65f71d51… | |

**readme.md**

## Continuous Integration server installation

This manual will let you install the whole continuous integration environment on a Mac.

### Requirements

- An Apple computer
- OS X Mavericks
- Access to the Ice network
- Internet, duhhhh

### Installation steps

1. Reserve an IP address and register a DNS name (ci-server) on the domain controller of IceMobile
2. Start up the fresh and clean Apple computer
3. Log on as the superadmin
4. Make an admin user account
    1. Go to `system preferences`
    2. Click on `user groups`
    3. Click the `+` button
    4. New account = administrator
    5. Full name = ci-server
    6. Account name = ci-server
    7. Password = whatever you like :)
5. Log out as `superadmin`
6. Log on as `ci-server`
7. Make sure the new user is logged on automatically
    1. Go to `system preferences`
    2. Click on `user groups`
    3. Click on `login option`
    4. Click on `automatic login`
    5. Choose `user logged in now`
8. Set the IP-address
    1. Go to `system preferences`
    2. Click on `network`
    3. Set the fixed IP address as the reserved address from step 1
9. Set power options
    1. Go to `system preferences`
    2. Click on `power options`
    3. Set `computer never sleeps`
    4. Set `start up automatically after power failure`
10. Turn off the screensaver
    1. Go to `system preferences`
    2. Click on `desktop and screensaver`
    3. Turn it off
11. Open the `Self service app` and install Xcode with it
12. Wait a bit :D

http://gitlab.ice.local/ben/ci-server-installation/tree/master Pagina 1 van 4

13. Open Xcode for the first time
14. Wait a bit more!
15. Install Homebrew, paste this in your terminal

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

16. Run brew doctor afterwards
17. Install NodeJS

```
brew install node
```

18. Install Jenkins

```
brew install jenkins
```

19. Follow the steps at the end of the installation to have Jenkins launched at login
20. Install Grunt-cli globally

```
npm install -g grunt-cli
```

21. Set up the Git connection for Gitlab
    1. Go to the terminal and do

    ```
    ssh-keygen -t rsa -C "user@domain.tld"
    ```

    2. Use default path
    3. Choose no password
    4. Then do this in the terminal

    ```
    cat ~/.ssh/id_rsa.pub
    ```

    5. Copy the whole output (starting with ssh and ending with the e-mail address)
    6. Go to the Ice GitLab server (http://gitlab.ice.local/)
    7. Logon with your LDAP credentials (must be someone with a access to all the repos that gonna be used on this server, the acces can also be added later on)
    8. Click on profile settings (top-right)
    9. Click on SSH keys
    10. Add a key
    11. Use title `ci-server`
    12. Paste the key (that is probably still on your clip-board, otherwise repeat substep 5 of step 22) in `key-field`
    13. Add key
22. Add the folder `~/repos`
23. In your terminal do:

```
cd ~/repos
git clone git@gitlab.development.ice.local:ben/ci-server-installation.git
cd ~/repos/ci-server-installations/files
cp .bash_profile ~/
cp homebrew.mxcl.jenkins.plist /usr/local/Cellar/jenkins/[current version]/
```

24. Download and install Google Chrome browser, use it when a browser is needed from now on
25. Download and install via the website of Java `JDK 7 u67`
26. Download via the website of Apache Ant `apache ant 1.9.4 binary packages`
27. Create the folder `~/developer`
28. Copy the downloaded Ant folder (apache-ant-1.9.4) to `~/developer`
29. Download via the website of Android `Android SDK Only for Mac` (you see it when you click on show all downloads on the bottom of the page)
30. Copy the downloaded Android SDK folder (android-sdk-macosx) to `~/developer`
31. Start a new terminal session
32. Then run this with terminal:

```
android
```

33. Install all android SDK tools from API 17 and higher

34. Install all android versions from API 17 and higher
35. Install all extras
36. Wait :) Get a coffee :)
37. Start jenkins with this command:

```
launchctl load ~/Library/LaunchAgents/homebrew.mxcl.jenkins.plist
```

38. Install Java SE 6 runtime when asked
39. Configure the security (configure this as you wish, but make sure `Anonymous` have at lesat the rights as mentioned in this substep)
    1. Go to http://localhost:8080 (http://localhost:8080)
    2. Go to `manage jenkins`
    3. Go to `configure global security`
    4. Enable security
    5. Security realm `jenkins own database`
    6. Allow users to sign up (this is for the admin account that we will create in a minute)
    7. Choose matrix based security
    8. Add `admin` user with all rights
    9. `anonymous` user needs the following rights: `overall read` , `job read, discover and workspace'
    10. Click `save`
    11. Create a new user
    12. Use the following options

```
username: admin
password: {what you like}
fullname: administrator
email: admin@icemobile.com
```

    13. Sign up!
40. Go to `manage jenkins`
41. Choose `configure system`
42. Number of executors must be `1`
43. Under jenkins location:

```
jenkins url: http://{dnsname}:8080/
email: admin@icemobile.com
```

44. Click the `save` button on the bottom!
45. Manage/install the plugins and jobs
    1. Go to `manage jenkins`
    2. Got to `manage plugins`
    3. Click on the `available` tab and install the following plugins:
        1. cucumber test result plugin
        2. git client plugin
        3. green balls
        4. managed script plugin
        5. xcode plugin
        6. git plugin (current version 2.2.5)
    4. Click on the `advanced` tab
    5. Upload `cucumber-reports.hpi` from the `ci-installation` repo
    6. Copy the folders from `jenkins-jobs` from the `ci-installation` repo to `~/.jenkins/jobs`
    7. Copy `buildstep-config-files.xml` from the `jenkins-scripts` folder from the `ci-installation` repo to `~/.jenkins/`
46. Restart Jenkins

```
launchctl unload ~/Library/LaunchAgents/homebrew.mxcl.jenkins.plist
launchctl load ~/Library/LaunchAgents/homebrew.mxcl.jenkins.plist
```

47. If there are jobs with GitHub repo's (for example iOS Jumbo atm), add the credentials for the server (ASK THE TEAM of that project for credentials!)
    1. Click on the job with the GitHub repo information (you need bo logged on again)
    2. Click on `configure`
    3. In the `SCM` block add you credentials (probably there is a keystring for the username and the password will be empty)
    4. Select the new credentials
    5. Click save!
48. Configure TrippleTest
    1. Run the job `TRIPPPLETEST_GIT_PULLER`
    2. Go to the workspace of this project, `~/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/`
    3. Run the following command in the terminal

```
node node_modules/appium/bin/appium-doctor
```

4. If asked for a Xcode select tools installation --> do it!!
5. And wait wait wait
6. When the installation is done continue `appium-doctor`
7. Everything should be green in the end
8. When this is not the case:
   - It is possible that you did not copy your `.bash_profile`
   - Or you did not start a new terminal session so your `.bash_profile` has not been loaded yet
   - Maybe the `.bash_profile` is outdated so check if it is correct, check the paths for Java, Android and Apache Ant
   - Make sure `appium-doctor` gives no errors in the end (use Google (http://www.google.com/) and the Appium documentation (http://appium.io/slate/en/master javascript#))
9. Authorize Appium to make use of `instruments` bye running the following in your terminal

```
sudo node node_modules/appium/bin/authorize-ios
```

10. Use your credentials when asked
11. When the iOS simulator is used for the first time, then fill in your credentials to authorize TripppleTest (just run a job that makes use of the iOS simulator)
12. Set the address of the Jira connector
    1. Open `~/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/config/support/jira.js`
    2. Set the property `connectorUrl`, use the DNS name
    3. Start the Jira-connector by running the following command from the terminal

```
cd ~/.jenkins/jobs/TRIPPPLETEST_GIT_PULLER/workspace/
grunt jira-connector
```

49. If it is O.K. you're done now. Configure the jobs as you want and use the current jobs as a template!

## Notes

- Make sure that all the connected devices trust the CI-Server
- If you use a physical iOS device make sure the certifications are installed properly
- Test your webhooks from GitLab and GitHub
- The TripppleTest framework can be found over here (http://gitlab.development.ice.local/evan/trippletest/tree/master)

## To-do

- Make a startup script for jira-connector
- Automatically restart the Jira-connector after an update of TripppleTest
- Let jobs run parallel
- Integrate Xcode 6 (waiting for an update on Appium, see here (https://github.com/appium/appium/pull/3517))
- Assert steps on design level (with screenshots perhaps)

# Bijlage VIII Handleiding testframework



| Name | Last Update | Last Commit › 1d79bb8054e (/evan/trippletest/commit/1d79bb8054e3acac230fb6789331ffedcbad034e) – updated android js | history (/evan/trippletest/commit |
|---|---|---|---|
| app (/evan/trippletest/tree/master/app) | 8 days ago | Waldo van Paddenburg (/u/waldo.vanpaddenburg) updated android js (/evan/trippletest/c… | |
| config (/evan/trippletest/tree/master/co… | 18 days ago | Rick de Hoop (/u/rick) commit for demo (/evan/trippletest/commit/fe9471b351d47f7c730f… | |
| docs (/evan/trippletest/tree/master/docs) | 2 months ago | Rick de Hoop (/u/rick) modified jenkins settings (/evan/trippletest/commit/08d4f5c28bff1c… | |
| lib (/evan/trippletest/tree/master/lib) | 18 days ago | Rick de Hoop (/u/rick) changed succesfull link from api to gui url (/evan/trippletest/commit… | |
| test (/evan/trippletest/tree/master/test) | 8 days ago | Waldo van Paddenburg (/u/waldo.vanpaddenburg) updated android js (/evan/trippletest/c… | |
| .gitignore (/evan/trippletest/blob/maste… | about a month ago | Rick de Hoop (/u/rick) fixed a bug with screenshotting on a failure (/evan/trippletest/com… | |
| Gruntfile.js (/evan/trippletest/blob/mast… | 23 days ago | Rick de Hoop (/u/rick) worked on the jira-connector, solved another bug from cucumberjs … | |
| package.json (/evan/trippletest/blob/m… | 19 days ago | Rick de Hoop (/u/rick) added jira support for CI environment, can be found in cucumber re… | |
| readme.md (/evan/trippletest/blob/mas… | 16 days ago | Rick de Hoop (/u/rick) removed some unwanted logs (/evan/trippletest/commit/5d90a391… | |

readme.md

## TripppleTest

TripppleTest is a framework for, primairy, automated acceptance tests. You can use this local on your own computer or put it in the continuous integration environment. You can more information about that over here (http://gitlab.development.ice.local/ben/ci-server-installation).

### Supported platforms

This framework uses Appium so it supports a few platforms out of the box.

- iOS
- Android
- FirefoxOS

See this page (https://github.com/appium/appium/blob/master/docs/en/appium-setup/platform-support.md) for the latest support information of Appium.

It can be used with other platforms also, just edit the before and after hooks in the 'test/support/setup.js' file.

### Installation and first test run

A few things need to be done before running your first test. Assuming you're on a Mac,

1. Install Xcode and open it for the first time
2. Install Homebrew, paste this in your terminal

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

3. Install NodeJS

```
brew install node
```

4. Install Grunt globally

```
sudo npm install -g grunt-cli
```

5. Get the TripppleTest framework

```
git clone git@gitlab.development.ice.local:evan/trippletest.git
```

6. Install the dependencies for the TripppleTest framework. Go into the TripppleTest framework folder and then

```
npm install
```

7. Check if you installed the Android SDK and Xcode correct. Go into the TripppleTest framework folder and then

```
node node_modules/appium/bin/appium-doctor
```

(make sure everything is checked for the platform you are gonna use)

Set up iOS for Appium (https://github.com/appium/appium/blob/master/docs/en/appium-setup/running-on-osx.md)

Set up Android SDK for Appium (https://github.com/appium/appium/blob/master/docs/en/appium-setup/android-setup.md)

8. Copy your mobile app, apk (for Android) or your app (for iOS), to the `/app` folder

9. Create a config file for your tests and save it in the `/config` folder

```
{
 "testSuite": "Jumbo", // Define the test-suite to be run when starting a test according this config-file
 "platformName": "iOS", // iOS, Android, or FirefoxOS
 "platformVersion": "7.1", // Specify the version (e.g. 8.0 or 4.4.2)
 "deviceName": "iPhone", // e.g. iPhone, iPad or Galaxy S4 (use auto for automatic Android lookup)
 "realDevice": "auto", // Use this property only if you want to run on a real iOS device (give UDID or auto)
 "appName": "Jumbo" // The name of the app to be tested without the extension
}
```

10. Write your feature file(s) and save them in the `/test/features/{testSuite}` folder
11. Implement the steps by creating the step files and save them in the `/test/steps/{platformName}/{testSuite}` folder (the name doesn't have to correspond with feature file name
12. Run the test according to your config file

```
CONFIG=configNameWithoutExtension grunt test --format pretty
```

or if you want to use the built-in appium inspector download and start the appium app from the website and run the test with the following command

```
CONFIG=configNameWithoutExtension grunt test-appium-local --format pretty
```

If you have followed all the steps and want to create new tests then go back to step 8. Happy testing!

## Examples

You find elements by using a subset of WebDriver's element-finding strategies. See finding elements (https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/finding-elements.md) for detailed information. We also have several extensions to the JSON Wire Protocol for automating mobile gestures (https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/touch-actions.md) like tap, flick, and swipe.

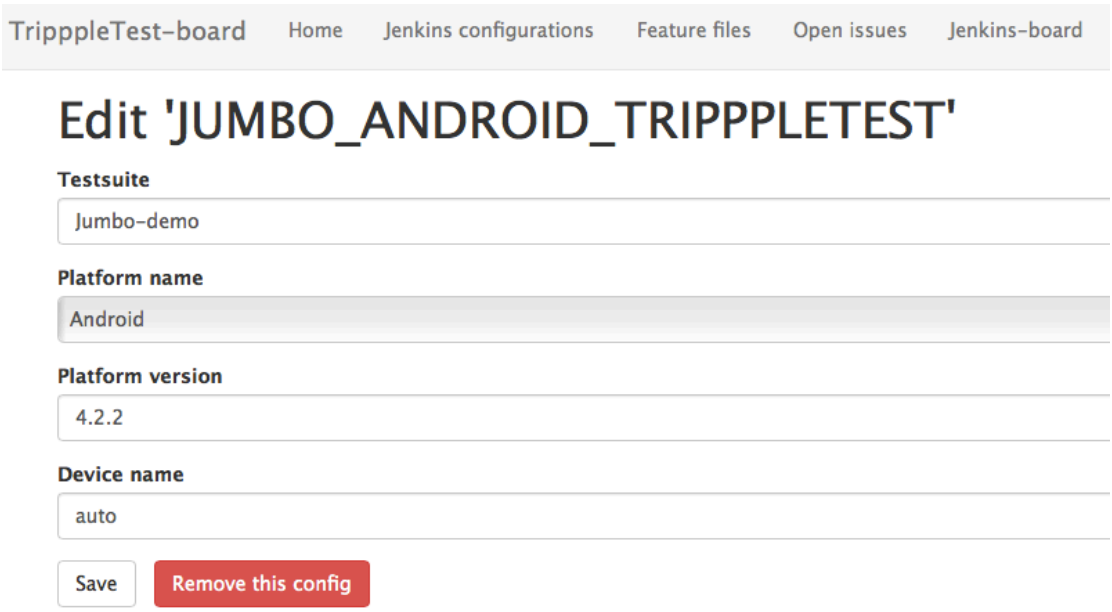
This repository contains many examples of tests in a variety of different languages (https://github.com/appium/sample-code)

For the full list of Appium doc pages, visit this directory (https://github.com/appium/appium/blob/master/docs/en)

You can also automate web views in hybrid apps! See the hybrid app guide (https://github.com/appium/appium/blob/master/docs/en/advanced-concepts/hybrid.md)

# Bijlage IX Screenshots dashboard

*De volgende twee screenshots zijn van een werkend dashboard dat niet is doorontwikkeld.*




# Bijlage X CucumberJS en Appium in actie