Professionele Bachelor Toegepaste Informatica

# Software delivery pipeline:

# Projectmanagement & projectkwaliteit

Stephanie Govaers

Promotoren:

Luca Filippone — Codrigo

Nele Custers — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

# Software delivery pipeline:

# Projectmanagement & projectkwaliteit

Stephanie Govaers

Promotoren:

Luca Filippone — Codrigo

Nele Custers — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

# Dankwoord

Dit eindwerk zou niet mogelijk geweest zijn zonder de hulp en steun van een aantal bijzondere personen.

In de eerste plaats wil ik mijn bedrijfspromotor, Luca Filippone, in het bijzonder oprecht bedanken voor de unieke kans die hij mij aangeboden heeft met deze uitdagende en mooie stageopdracht. Gedurende het stageverloop heeft hij steeds klaargestaan met waardevol advies, zowel voor de stage als daarbuiten. Hij heeft voor mij echt de rol van mentor opgenomen, en ik kan oprecht zeggen dat hij mij in mijn professionele vaardigheden en als mens heeft laten groeien.

Ik bedank ook graag mijn hogeschoolpromotor, Nele Custers, voor haar onuitputtelijk enthousiasme en hulp in het voltooien van dit eindwerk. Zij was altijd bereikbaar en zocht altijd actief mee naar oplossingen van problemen die op mijn pad kwamen.

Verder bedank ik ook graag het hele Bewire-team. Vanaf de eerste dag heeft iedereen mij ontzettend welkom laten voelen. Als stagiaire hebben ze mij altijd als *part of the team* behandeld; zowel op de werkvloer als op de talrijke Bewire-evenementen daarbuiten. Bovendien stonden tal van specialisten steeds klaar om met hun unieke vakkennis mijn eindwerk te verrijken.

Ook de andere stagiairs ben ik een bedankje schuldig, met mijn collega, Mathias Grauwels, in het bijzonder. Hij heeft mij altijd ontzettend gesteund en ik weet dat ik aan hem een vriend voor het leven heb na het afronden van deze stage.

Ik bedank ook alle lectoren en het departementshoofd van PXL-IT om mij te vormen tot de softwaremanager die ik nu ben. In het bijzonder wil ik alle SWM-lectoren bedanken voor hun ongelooflijke inzet en menselijke aanpak, zowel voor hun respectievelijke vakinhoud, als voor de studenten zelf.

Ik wil ook mijn moeder bedanken. Zij is zonder twijfel verantwoordelijk voor al mijn verwezenlijkingen. Als alleenstaande moeder heeft zij mij geleerd hoe sterk een vrouw kan zijn en ondanks alle tegenslagen kan overwinnen. Zij heeft mij altijd het goede voorbeeld gegeven.

Tot slot wil ik ook mijn vriend, Dennis, bedanken om mijn steun en toeverlaat te zijn in elk moment van twijfel, om mijn angsten weg te nemen en om altijd trots op mij te zijn.

# Abstract

Uit een partnership tussen Codrigo, het stagebedrijf, en zusterbedrijf Evance is een start-up ontstaan, die de naam “Appmind” draagt. Binnen Appmind worden kwaliteitsvolle, cross-platform, mobiele applicaties ontwikkeld.

Aangezien het om een start-up gaat, moet een geschikte ‘software delivery pipeline’, die aansluit bij de noden en visie van Appmind, omschreven worden. Hiervoor is een stageopdracht uitgeschreven, waarin Codrigo de studenten uitdaagt om een krachtige pipeline te definiëren en gedeeltelijk te implementeren. Enerzijds ligt hier de focus op de ideale tools, maar daarnaast moeten ook de nodige processen zorgvuldig geschetst worden.

Binnen de Bewire-groep is al heel wat kennis en ervaring opgedaan in het ontwikkelen van applicaties. De uitdaging van de stageopdracht ligt dan ook in het analyseren van de huidige situatie binnen Bewire, om vervolgens een beslissing te maken om bestaande praktijktoepassingen van de Bewire-groep al dan niet te adopteren, te verbeteren, uit te breiden of geheel nieuwe tools te selecteren.

De focus van het eindwerk zijn tools en processen die projectmanagement en projectkwaliteit bevorderen, zoals *Atlassian Lifecycle Management.*

Binnen het onderzoek wordt een antwoord gegeven op de vraag “Hoe kunnen DevOps-principes ingezet worden om de kwaliteit van projectmanagement en de projectkwaliteit te verbeteren in kortlopende projecten voor mobile development?” DevOps wordt immers vaak uitsluitend technisch geïnterpreteerd. Daarnaast worden de voordelen vaak enkel beschreven vanuit het oogpunt van het development-team of het operations-team. Het onderzoek beschrijft concrete scenario’s en toepassingen van DevOps voor projectmanagers.

# Inhoudsopgave

# Lijst van gebruikte figuren

# Lijst van gebruikte afkortingen

| AFKORTING | VERKLARING |
|---|---|
| ABC | Assignments, Branches, Conditionals |
| API | Application Programming Interface |
| AVG | Algemene Verordening Gegevensbescherming |
| BDD | Behavior-Driven Development |
| BPMN | Business Process Model Notation |
| CI | Continuous integration |
| CRM | Customer Relationship Management |
| CSS | Cascading Style Sheets |
| DPO | Data Protection Officer |
| ERD | Entity Relationship Diagram |
| GDPR | General Data Protection Regulation |
| HTML | HyperText Markup Language |
| ID | Identificatienummer |
| IDE | Integrated Development Environment |
| iOS | iPhone Operating System |
| ITIL | Information Technology Infrastructure Library |
| JQL | Jira Query Language |
| JSON | JavaScript Object Notation |
| MDM | Mobile Device Management |
| OAuth2 | Open Authorization |
| OWASP | Open Web Application Security Project |
| PaaS | Platform as a Service |
| POM | Project Object Model |

| | |
|---|---|
| QA | Quality Assurance |
| RACI | Responsible, Accountable, Consulted, Informed |
| SLA | Service Level Agreement |
| SWOT | Strengths, Weaknesses, Opportunities, Threats |
| TDD | Test-Driven Development |
| TM4J | Test Management for Jira |
| UAT | User Acceptance Tests |
| UI | User Interface |
| UML | Unified Modeling Language |
| URL | Uniform Resource Locator |
| VM | Virtual Machine |
| XP | Extreme Programming |

# Inleiding

In het kader van een nieuwe start-up, binnen Bewire, die de naam Appmind draagt, heeft Codrigo de studenten uitgedaagd om een nieuwe *software delivery pipeline* te omschrijven. Deze *pipeline* moet naast de gepaste *tools* ook de bijhorende processen definiëren. Binnen Appmind ligt de focus op het ontwikkelen van *cross-platform mobile applications.* Deze *development* projecten zijn vaak van kortlopende duur. Daarom is het belangrijk om op zoek te gaan naar een *pipeline* die dit doel ondersteunt.

Binnen dit eindwerk wordt de focus gelegd op de producten van Atlassian en dan voornamelijk Jira. Er wordt een diepgaande automatisatie beschreven voor bijvoorbeeld het verplaatsen en beheren van *issues.*

In het onderzoekstopic wordt dan een antwoord gezocht op volgende onderzoeksvraag: “Hoe kunnen DevOps-principes ingezet worden om de kwaliteit van projectmanagement en de projectkwaliteit te verbeteren in kortlopende projecten voor mobile development?”

Met de DevOps-principes kan de nood van Bewire en Appmind immers perfect beantwoord worden.

# I. Stageverslag

## 1 Bedrijfsvoorstelling

Codrigo profileert zich als *software delivery competence center* binnen de Bewire-groep. Projectmanagers gaan er aan de slag om uiteenlopende projecten tot een goed einde te brengen, rekening houdend met alle facetten van softwaremanagement en *agile* principes. Alle stappen van de *software development* cyclus worden doorlopen, steeds met het oog op de businessdoelen van de klant. Zo gaat een projectmanager binnen Codrigo niet enkel bepalen wat de technische *requirements* zijn van een applicatie, maar ook hoe een applicatie aansluit bij de *business needs* en toekomstvisie van de vragende partij. De resultaten van deze analyses worden dan gedefinieerd en gedocumenteerd, steeds met terugkoppeling naar de *product owner.* Vervolgens worden de applicaties dan gebouwd in een agile context met het oog op een snelle *time to market* en de hoogste kwaliteitsnormen.

Codrigo scoort ook hoog op flexibiliteit. Zo kunnen projecten *in-house* aangepakt worden, maar ook *embedded* binnen het bedrijf van de vragende partij, of zelfs op gecombineerde locaties. Codrigo richt zich immers op een diepgaande en waardevolle samenwerking met haar klanten.
Naast het succesvol doorlopen en opleveren van projecten, ondersteunt Codrigo ook lopende projecten die bijvoorbeeld niet meer *on track* zijn, of buiten scope of budget dreigen te gaan. Terwijl bedrijven vaak onnodig de reflex hebben om meer *developers* op een dergelijk project te zetten, gaat Codrigo daarentegen op zoek naar innovatieve, agile oplossingen om een project terug on track te krijgen.

Hoewel de stage doorgaat in naam van Codrigo, toch is het belangrijk om ook even de activiteiten van Evance NV onder de loep te nemen, aangezien uit beide bedrijven een partnership is ontstaan, waaruit een start-up is voortgekomen; Appmind. Alle stageactiviteiten zullen in dienst staan van de ontwikkeling van de *software delivery pipeline* van Appmind, maar daarover later meer.


*figuur 1 - bedrijfstructuur binnen Bewire*

Evance, dat eveneens deel uit maakt van de Bewire-groep, richt zich op het ontwikkelen van frontend oplossingen, zowel voor web- als mobile toepassingen. Evance werkt eveneens op basis van consultancy.

Tot slot is ook enige kadering nodig voor de start-up waarvoor het onderzoek uitgevoerd zal worden. Appmind is een bedrijf dat zich specialiseert in het ontwikkelen van mobiele applicaties. Het bedrijf differentieert zich van de grote massa door enerzijds de applicaties cross-platform op te leveren, en zet daarnaast sterk in op *user experience.* Opnieuw vormt een intensieve samenwerking met de klant en maatwerk de rode draad in het ontwikkelingsverhaal, zoals al gebleken is uit bovengenoemde bedrijven binnen Bewire.

# 2 Voorstelling stageopdracht

## 2.1 Probleemstelling

Zoals in voorgaande punten beschreven, is uit het partnership tussen Codrigo en Evance een start-up ontstaan, die de naam Appmind draagt. Binnen Appmind zullen kwaliteitsvolle, cross-platform applicaties ontwikkeld worden.
Aangezien het om een start-up gaat, hoort de ideale *software delivery pipeline*, met de nodige flow van processen en ondersteunende *tools* nog onderzocht en omschreven te worden.
Hoewel er binnen de Bewire-groep al veel kennis en ervaring is opgedaan met het ontwikkelen van applicaties, is het belangrijk dat we een selectie gaan maken op basis van de bedrijfsbehoeften en visie van Appmind. We kunnen dus bestaande praktijktoepassingen van de Bewire-groep adopteren en verbeteren of uitbreiden waar nodig, of geheel nieuwe tools selecteren.
Het is dus belangrijk om eerst de business needs in kaart te brengen. Daarna wordt nagegaan hoe de *software delivery flow* momenteel wordt geïnterpreteerd binnen Codrigo en Evance, om vervolgens de *software delivery pipeline*, met gepaste *tools* en processen en aansluitend met de toekomstvisie van Appmind te definiëren.

## 2.2 Aanpak

### 2.2.1 Vastleggen van de business needs & requirements

Een eerste stap is uiteraard het proberen vastleggen van de requirements en het afbakenen van de *scope* van het project. Hiervoor is een intakegesprek nodig met de bedrijfspromotor en andere stakeholders binnen Appmind. Hierbij kunnen we eventueel business en system usecases opstellen, om al erg snel te concretiseren. Zo verminderen we eveneens de kans dat het onderzoek steunt op foutieve veronderstellingen.
Naast het proces van *requirements elicitation*, is het ook noodzakelijk om de businessdoelen van Appmind onder de loep te nemen. We willen immers een *pipeline* bouwen die werkt voor intern gebruik, maar we willen ook dat de *pipeline* een middel is om de externe doelen te behalen.
In een eerste fase van het project stel ik daarom voor, als er al een interne en externe analyse is gebeurd voor Appmind, om een analyse van de *Strengths, Weaknesses, Opportunities en Threats (*SWOT-analyse) voor de organisatie te maken. Op die manier kunnen we een goed beeld schetsen van de bedrijfsdoelen van Appmind en welke kansen en bedreigingen daarin een rol spelen enerzijds, en welke sterktes en zwaktes intern zijn anderzijds. Op die manier kunnen we strategische keuzes analyseren en formuleren tijdens ons onderzoek naar de best mogelijk *software delivery pipeline.*

### 2.2.2 Analyseren van de bestaande situatie

In een volgende fase wordt er gedocumenteerd welke tools en processen momenteel deel uit maken van de *software delivery pipeline* binnen Bewire. Op deze manier kunnen we nagaan welke methodieken en tools behouden en/of verbeterd kunnen worden, en waar volledig nieuwe implementaties mogelijk zijn. Bovendien is het ook belangrijk om te weten hoe een projectteam binnen Appmind samengesteld is. Mogelijke methodieken die we hier kunnen gebruiken zijn het opstellen van een *Responsible-Accountable-Consulted-Informed-schema (*RACI) om de verantwoordelijkheden van het projectteam te zien. Meestal is er geen sprake van een one-size-fits-all aanpak voor alle projecten, maar het is zinvol om een algemeen beeld te schetsen van de interne werking binnen de huidige projecten van Bewire.

### 2.2.3 Het uitvoeren van het onderzoek

Vervolgens gaat het hoofdproces van het onderzoek van start, namelijk het onderzoeken en definiëren van een software delivery pipeline, met de gepaste tools en processen. Naast een literatuuronderzoek, is het dus belangrijk om voldoende casestudies uit te voeren.
Bovendien zijn feedback loops cruciaal voor dit onderzoek. Er wordt immers steeds meer kennis opgedaan, die dan weer kan afgestemd worden op de vragen en verwachtingen binnen Appmind. Zoals omschreven in de stageopdracht worden de volgende vragen beantwoord:

*"Hoe kunnen we Jira in dit process integreren? Welke mogelijkheden zijn er om snel kleine projecten op te zetten in Jira?"*
*Hoe kunnen we uren laten loggen door ontwikkelaars en deze linken aan budget? Kan dit rechtstreeks in Jira? Hebben we plugins nodig? Hoe deze configureren?*
*Met welke tooling gaan we kwaliteit meten? Hoe? Hoe integreert dit? Met codebase of met Jira?*
*Vanaf dag 1 zijn er omgevingen nodig? Kiezen we voor AWS? DigitalOcean? Waarom? Pricing?*
*Hoe en op welk punt gaan we continuous integration gaan toepassen?*
*Wat met cross platform testing? Kunnen we UI-automatisatie testen laten lopen via een cloud systeem? Hoe gaat dit in zijn werk? AWS Appfarm?*
*Hoe omgaan met een testomgeving? HockeyApp gebruiken om test deploys & distribution te doen of iets anders? Waarom?*
*Deployment naar de stores, hoe kan dit zo vlot mogelijk gebeuren?*
*Wat met support naar klanten achteraf? Hoe kunnen klanten support tickets gaan loggen en kunnen wij deze gestroomlijnd opnemen? Servicedesk? Zendesk?*
*Gaat elke klant een support omgeving krijgen? Werkt dit met accounts?*
*Kan je zelf een flow aanmaken om incidenten te gaan 'filteren'?*
*Hoe kunnen we het support process inhaken op ons development process? M.a.w. support tickets koppelen aan Jira tickets.*
*Business apps die niet publiekelijk zijn deployen naar een corporate app store, wat komt hierbij kijken? Hoe werkt dit licentiemodel? Wat zijn de kosten?*
*GDPR? Moeten we ergens rekening mee houden?"* [1]

Het onderzoek kan onderverdeeld worden in zes grote onderzoeksdomeinen. In elk onderzoeksdomein wordt afwisselde onderzoek gedaan, worden verschillende tools en plug-ins getest en worden de flow en onderdelen van de *pipeline* opgezet. Deze onderzoeksdomeinen zijn Atlassian *lifecycle management, security & quality, cloud hosting, cross platform testing, deploy & licenses* en tot slot *support platforms*.

#### Atlassian Lifecycle management

Binnen dit onderdeel wordt de werking van Jira onder de loep genomen. Hoe kunnen bepaalde *flows* snel opgezet en hergebruikt worden. Hoe kunnen we *workload* gaan tracken? Hoe kunnen we effectief kosten bijhouden per project en kunnen we opbrengsten berekenen per project? Welke *tools* zijn daarvoor nodig? Kunnen we deze financiële gegevens ook visualiseren? Kunnen we deze data extraheren?

#### Security & quality

Aan welke kwaliteitsnormen moeten mobiele applicaties voldoen? Hoe kunnen we die kwaliteit meten en aantonen? Welke *code quality tools* zijn aangeraden? Welke security-maatregelen kunnen we nemen, los van de code? Welke maatregelen moeten we nemen voor *iPhone Operating System* (iOS-)applicaties en welke voor Android? Hoe kunnen we functionele testen integreren in Jira?

#### Cloud hosting

Welk *cloud hosting platform* is aangeraden? Welke verschillen zijn er tussen alle *tools*? Gebruiken we misschien beter een *Platform as a Service Provider* (PaaS)? We kunnen hier een analyse maken van enerzijds de noden en anderzijds van de kosten. Wat moet er gebeuren om *GDPR-compliant* te zijn? Welke contracten moeten hiervoor afgesloten worden? Wat zijn de mogelijkheden naar support toe, zowel voor het stagebedrijf zelf, als voor klanten. Welke kosten zijn aan support verbonden? Welke *push notification serviceproviders* zijn er? Wat zijn de verschillen? Werken die cross-platform? Welke kosten zijn daaraan verbonden? Welke adviezen hebben wij?

#### Cross platform testing

Welke testplatformen zijn er en wat zijn de verschillen? Hoe kunnen we cross-platform gaan testen? Welke kennis is daarbij vereist? Zijn deze tools en hun configuratie voldoende gebruiksvriendelijk? Is er een *change* nodig in de code? Wat kan er allemaal getest worden? Hoe kunnen resultaten gerapporteerd worden? Hoe kunnen we dit integreren binnen Jira? Kunnen we via een bepaalde service een script runnen om onze database te wissen?

#### Deploy & licenses

Hoe kunnen we de app verspreiden voor testpubliek? Welke platformen bestaan hiervoor? Wat kunnen deze platformen nog? Kunnen we ook een platform gebruiken om de app naar de officiële store te pushen via een geautomatiseerd proces? Welke apps mogen in welke stores, rekening houdend met de verschillende licenties? Wat kunnen *Mobile Device Management* (MDM) tools vandaag om apps op bedrijfsniveau uit te rollen? Welke restricties kunnen we hier opleggen? Hoe werkt dit proces? Welke kostenplaatje is hieraan verbonden?

#### Support platforms

Welke platformen zijn aangraden? Op welke manieren kunnen klanten tickets loggen en hoe kunnen we dit op een platform integreren? Kunnen we meerdere klanten aan een support desk linken? Hoe werkt dit? Welke flows zijn er voor tickets? Hoe kunnen we aan rapportage gaan doen? Bijvoorbeeld, hoeveel issues komen er binnen per applicatie? Hoe lang duurt het voor een issue wordt opgenomen? Hoe gaan we om met *Service Level Agreements (*SLA)? Hoe kunnen we een helpdesk integreren in Jira? Hoe kunnen we dit proces laten inhaken op het *development proces*?
Merk op dat onderzoek mogelijk zal tonen dat de relevantie van bepaalde onderzoeksdomeinen verschillend is, en dat bijgevolg meer aandacht en tijd zal geschonken worden aan deze onderzoeksdomeinen. Bovendien zal er doorheen het proces ook steeds de link gelegd moeten worden tussen elk van deze onderzoeksdomeinen. Tijdens dit onderzoek is het dus erg belangrijk om aan de hand van feedback voldoende bij te sturen.

### 2.2.4 Formuleren van de software delivery pipeline.

De uiteindelijke selectie van *tools* kan opgeleverd worden in een vergelijkingsmatrix, terwijl de processen kunnen uitgeschreven worden in een handleiding en eventueel kunnen aangevuld worden met een *Business Process Model Notation* (BPMN) of andere schematische voorstelling voor de gehele *pipeline*.

# 3 Uitwerking stageopdracht

## 3.1 Atlassian Lifecycle Management

Zoals eerder beschreven, wordt binnen dit onderdeel de werking van Jira en andere Atlassian tools onder de loep genomen. Onderwerpen die aan bod zullen komen zijn bijvoorbeeld *workflows* binnen Jira, een onderzoek naar de ideale *roadmap*-tool, het loggen van uren, GDPR-compliancy binnen de Atlassian suite en tot slot synchronisatie tussen verschillende Jira-projecten en zelfs tussen verschillende Jira-instanties.

### 3.1.1 Jira workflows

#### 3.1.1.1 Inleiding

Jira *workflows* worden gedetailleerd uitgelegd in mijn onderzoek. Lees *"DevOps: Een theoretisch kader*

*Wat* is DevOps?

Vooraleer praktische implementaties van DevOps in het kader van projectmanagement en projectkwaliteit omschreven worden, is het noodzakelijk te begrijpen wat DevOps eigenlijk is.

DevOps is een samenvoeging van het woord '*development'* en het woord '*operations.'* DevOps is een methode om op een *agile* manier software te ontwikkelen, maar gaat eigenlijk nog een stap verder dan "*agile."* Het gaat bij DevOps immers om de samenwerking tussen de softwareontwikkelaars en het *operations*-team. Beiden zijn immers verantwoordelijk dat het gewenste eindresultaat opgeleverd kan worden. In feite integreert DevOps dus beide teams om samenwerking en productiviteit te verbeteren. Naast deze samenwerking, zijn er nog een aantal principes die gebonden zijn aan DevOps en die in het volgende onderdeel toegelicht worden.

## 3.2 Welke DevOps-principes zijn er?

Er bestaan heel wat definities van DevOps, maar toch blijft de essentie van DevOps vaak open voor interpretatie. Vandaar dat er vaak gewerkt wordt met een aantal principes die eigen zijn aan DevOps, om het begrip te kunnen kaderen. Hoewel er ook geen consensus is over welke principes DevOps definiëren en veel bedrijven dus een eigen invulling geven aan DevOps, zijn er een aantal principes die in deze paper weerhouden worden, namelijk:

a) Accountability, ownership & end-to-end responsibility
b) Collaboration
c) Automation
d) Continuous integration
e) Fast & reliable deploys with continuous delivery
f) Continuous Improvement

**Accountabililty, ownership & end-to-end responsibility**

Met *accountability* en *end-to-end responsibility* wordt doorgaans bedoelt dat zowel *development* als *operations* een gedeelde verantwoordelijkheid hebben tijdens het gehele *software delivery proces.* Dit betekent dat beide teams een idee hebben van wat er allemaal speelt tijdens elke fase van de

*software development life cycle.* In het kader van dit onderzoek wordt dit kader zoveel mogelijk verbreedt, namelijk om niet enkel de ontwikkelaars of *operation engineers* in deze verantwoordelijkheid te betrekken, maar ook andere profielen, zoals projectmanagers en testers. Uiteraard is het niet de bedoeling om de taaklast van elk teamlid uit te breiden, of om bijkomende verantwoordelijkheden op elk individu te storten, maar wel om

### Collaboration

Binnen DevOps is het concept van *collaboration* of samenwerking cruciaal. Het team komt samen om problemen van eender welke aard aan te pakken. Het is dus belangrijk dat er gepraat wordt en dat er begrip wordt opgebracht voor ieders verantwoordelijkheden, taken, enzovoort. Er wordt zo transparant mogelijk gewerkt, en er wordt actief ingezet om de betrokkenheid binnen het team te vergroten.

### Automation

DevOps is gericht op snelle opleveringen. Het is dus logisch om zoveel mogelijk te gaan automatiseren. Hoe meer handelingen automatisch gebeuren, hoe minder menselijke input nodig is. Dat betekent dat er enerzijds minder kans is dat menselijke fouten in het proces geïntroduceerd worden en dat anderzijds de werklast van de teamleden verminderd. Zij kunnen hun tijd besteden aan andere aspecten van hun job, waardoor er meer werk gedaan kan worden, en waardoor hopelijk de tevredenheid van de werknemers verhoogt.

### Continuous Integration (CI)

Met CI wordt de gebouwde code getest. Zie het hoofdstuk over CI met Bitbucket Pipelines voor een meer volledige omschrijving van CI en een praktische uitwerking met binnen de Atlassian stack.

### Fast & reliable deploys with Continuous Delivery (CD)

Na CI is CD de volgende stap. CD is de mogelijkheid om *changes*, ongeacht hun aard, snel in productie of rechtstreeks in de handen van de gebruikers te krijgen. Bovendien moet dit op een snelle, veilige en duurzame manier. Het doel is dus het optimaliseren van *deployments.* Het is dus een vereiste dat de code steeds in een *deployable* state is, zelfs al worden dagelijks talloze veranderingen aan de code toegevoegd.
Hoewel projectmanagers onrechtstreeks baat kunnen hebben bij CD, ligt de nadruk van CD voornamelijk op *deployment.* Vandaar dat een toepassing van CD voor het bevorderen van projectmanagement als *out of scope* beschouwd worden binnen deze paper.

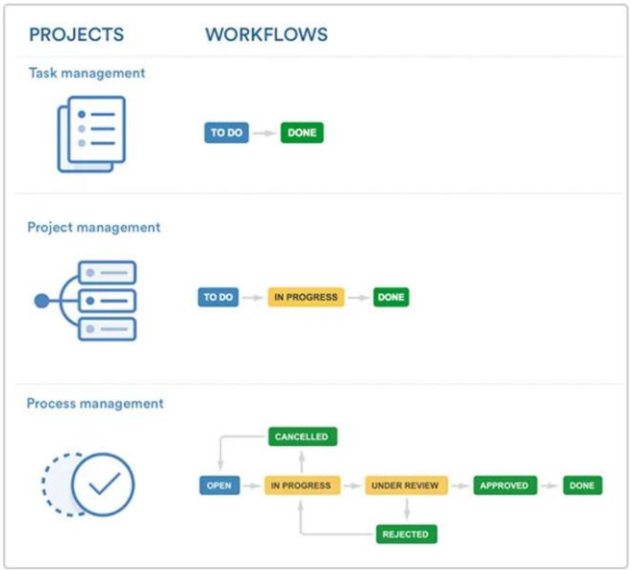
### Continuous Improvement

Veel organisaties die met DevOps willen beginnen, trappen in de valkuil dat bovenstaande principes allen onmiddellijk en volledig geïmplementeerd moeten worden. Dit is echter niet het geval. Een DevOps-cultuur moet steeds pragmatisch bekeken worden binnen een bedrijf. Welke gedachtegangen, middelen of methodieken zijn zinvol voor de organisatie in kwestie? Bovendien worden organisaties ook voortdurend geconfronteerd met veranderingen. De noden van klanten veranderen, de bestaande technologieën vernieuwen, de maatschappij evolueert, enzovoort. DevOps is een praktijk die zich steeds moet blijven aanpassen en inspelen op de noden van de organisatie en de maatschappij daarrond. Daarom is het belangrijk dat er voortdurend gezocht wordt naar mogelijke verbeteringen en manieren om te innoveren.

DevOps toegepast binnen de Atlassian suite*”* binnen het onderzoek voor een meer uitgebreide uitleg over de werking van *workflows* en mogelijke DevOps-toepassingen.

#### 3.2.1.1 Probleemstelling

Bij de aanvang van een nieuw project, gaat er steeds heel wat werk aan het eigenlijke *development*-proces vooraf. Denk bijvoorbeeld aan het bepalen van de *requirements*, het opstellen van een *backlog*, enzovoort. Ook binnen Jira kan het nodige configuratiewerk noodzakelijk zijn. Afhankelijk van de projecteisen, kan een op maat gemaakte *workflow* binnen Jira een absolute meerwaarde zijn. Denk hier bijvoorbeeld aan het toevoegen van automatische transities, een vaste workflow voor het *Quality Assurance team* (QA-team) terwijl ook het *development team* een eigen standaard heeft, enzovoort. Een absolute *must* binnen dit onderzoeksdomein is het omschrijven van een gepaste manier om *workflows* op te zetten, zodat dit binnen projecten kan gereproduceerd worden.




*figuur 2 - Jira workflows voor verschillende doeleinden*

#### 3.2.1.2 Uitwerking

Mijn vertrekpunt was het opzetten van een eigen *workflow* voor een project. Vervolgens is gecontroleerd hoe een *workflow* gekoppeld kon worden aan nieuwe projecten.

In het kader van het onderzoek, is vervolgens onderzocht op welke manieren het huidige gebruik van *workflows* binnen Bewire verbeterd zou kunnen worden, vanuit een DevOps-optiek. Deze uitgebreide verslaggeving kan gevonden worden binnen de uitwerking van het onderzoek.

Samengevat zijn volgende toepassingen gerealiseerd:

g) Het automatiseren van transities via *workflow triggers*
h) Het automatiseren van allerlei taken via een plug-in
i) Continuous Integration met Bitbucket Pipelines

**Het automatiseren van transities via *workflow triggers***

Een absolute meerwaarde voor Bewire, en vooral voor bedrijven zoals Appmind, waar snelheid centraal staat, is het inbouwen van automatisatie binnen Jira. Concreet betekent dit dat *issues* automatisch van de ene status naar de andere verplaatst worden op basis van een *trigger.* Zo kan een issue bijvoorbeeld automatisch in de status '*In Progress'* of '*In Development'* geplaatst worden wanneer een *developer* een *feature branch* in de *repository* aanmaakt vanuit Jira, of wordt een issue naar *done* verplaatst wanneer een *pull request gemerged* wordt op de *development branch*.

Dit wordt gerealiseerd door *triggers* toe te voegen op *transitions* tussen bepaalde statussen in de *workflow*.

### Het automatiseren van allerlei taken via een plug-in

Wanneer er al automatisatie via *workflow triggers* geconfigureerd is, zal de flow binnen projecten zeker al heel wat vlotter verlopen. Toch zijn er enkele toepassingen die niet via een workflow geautomatiseerd kunnen worden. Zo is het bijvoorbeeld niet mogelijk dat het verplaatsen van subtaken van de ene status naar de andere, geen invloed heeft op de corresponderende hoofdtaak. Uiteraard willen we dat hoofdtaken op een juiste manier verplaatst worden wanneer we werken met automatische taken. Moest dit niet het geval zijn, dan zouden gebruikers nog steeds evenveel tijd moeten steken in het controleren en manueel up-to-date houden van issues, en dan zou de automatisatie haar doel voorbijgaan.

Via de plug-in *Automation* zijn tal van automatisaties gerealiseerd:

- Het synchroniseren van subtaken en hoofdtaken aan de hand van een aantal regels
- Het automatisch toekennen van *issues* aan een ontwikkelaar
- Het verwittigen van een *senior developer* of projectmanager als er openstaande *pull requests* zijn
- De projectmanager en toegekende Jira-gebruikers notificeren van openstaande *high-priority issues* via Slack
- Alle subtaken van één *parent-issue* naar de finale status verplaatsen

### Continuous Integration met Bitbucket Pipelines

Voor een meer complete integratie binnen de Atlassian-kit, zijn de mogelijkheden met een Bitbucket-integratie onderzocht. Bitbucket werd al gebruikt binnen Bewire, en is zeker een meerwaarde, aangezien Bitbucket ook een product van Atlassian is, waardoor de integratie eenvoudig op te zetten is, en een aantal wenselijke functionaliteiten voorzien worden.

In het kader van het onderzoek zijn mogelijke verbeteringen onderzocht, opnieuw met het oog op een DevOps-implementatie.

Aangezien er al een volledige automatisatie voorzien was binnen de workflow van een project, die dan ook nog eens berustte op de werking van de gebeurtenissen binnen de *repository* (bijvoorbeeld

het verplaatsen van issues bij het aanmaken van een feature-branch), was het opzetten van *Continuous Integration* (CI) een logische volgende stap.

Hiervoor is gewerkt met Bitbucket Pipelines. Allereerst is een Java-repository opgezet, gebaseerd op een Maven-project. In dit eenvoudige project is enerzijds een hoofdklasse geschreven, en anderzijds enkele *unit tests* die de methodes uit de hoofdklasse gaan testen. Vervolgens is een script geschreven om de *pipeline* te initialiseren, de *repository* te builden en vervolgens de *unit tests* automatisch te laten lopen. Deze werking gebeurt immers elke keer een *developer* een *pull request* om naar de *development-branch* te *mergen* indient.

Door deze implementatie is CI gerealiseerd. Hierdoor kan vermeden worden dat niet-werkende code toegevoegd wordt aan de *development-branch en* dat er veel tijd moet gestoken worden in het herstellen van *broken builds.* Bovendien kan code sneller in productie en kunnen *developers* efficiënter werken. Tot slot zal de uiteindelijke productkwaliteit hoger liggen.

### 3.2.2 Roadmap tool

#### 3.2.2.1 Inleiding

Een roadmap is een *high-level planning*, waarop alle belangrijke mijlpalen van een complex project visueel worden weergegeven. Naast de mijlpalen worden ook verbanden tussen verschillende deelprocessen vastgelegd. Een roadmap kan een ideaal middel zijn om een planning te communiceren naar klanten of andere stakeholders van een project. [2]

#### 3.2.2.2 Probleemstelling

Binnen Bewire worden roadmaps gebruikt als communicatiemiddel naar klanten en stakeholders toe. Het is bovendien een krachtig middel om projecten op te volgen en te beheren. In de praktijk is het echter zo dat de *scope* van een project wel eens durft te veranderen, waardoor ook de planning moet aangepast worden. De roadmaps die binnen Bewire gemaakt werden, waren niet gekoppeld aan de *backlog* van een bepaald project. Aangezien de gecentraliseerde informatie dus niet gekoppeld was aan de geproduceerde roadmap, moesten de roadmaps telkens manueel aangepast worden door een veranderende *scope*. Dit is een tijdrovend proces, dat bovendien gevoelig is aan menselijke fouten. Denk bijvoorbeeld aan het inschatten van een precieze einddatum van het project, of correct inschatten van de capaciteit van ingezette werknemers. Daarnaast is het ook moeilijk om manueel te controleren of bepaalde werknemers niet overboekt zijn, zeker wanneer een werknemer op meerdere projecten wordt ingezet. Dit probleem zou opgelost kunnen worden door een geschikte roadmap-plug-in te installeren, die meteen ook gekoppeld is aan issues binnen Jira. Op deze manier kunnen tickets immers *live* opgevolgd worden, en kan de roadmap dynamisch aangepast worden op basis van de ontwikkelingen binnen het gekoppelde project.

De eisen voor de gewenste tool zijn weergegeven in onderstaande MoSCoW-matrix.

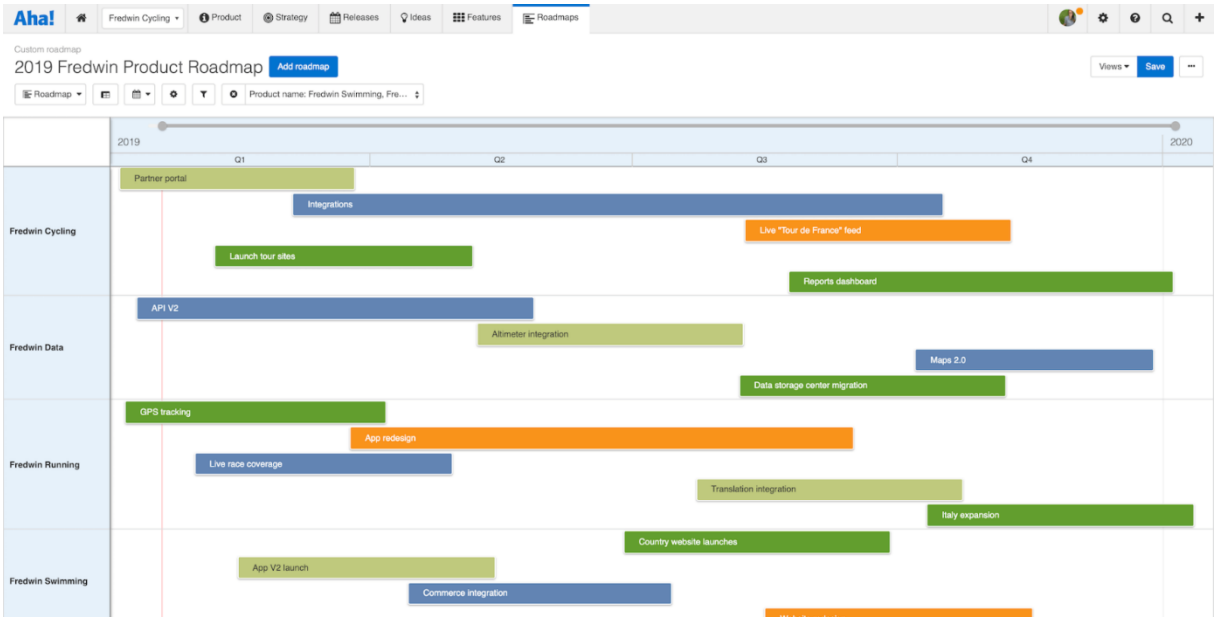
| Must have | Should have | Could have | Won't have |
|---|---|---|---|
| Duidelijk genereerbare rapporten voor stakeholder-communicatie | Epic-level integratie | Gantt-chart | Plug-ins van derden |
| Jira-integratie | Resource management | Risico-analyse | |
| OAuth2 authenticatie voor gebruikers | Redelijke prijs | | |

*figuur 3 - MoSCoW-matrix van requirements*

### 3.2.2.3 Mogelijke tools

#### Aha!

Aha! is een softwarepakket dat als vrijstaand product kan aangeschaft worden. Daarnaast kan Aha! ook via de corresponderende plug-in geïntegreerd worden met Jira. Naast roadmaps biedt Aha! een volledig pakket van tools aan waarbij tal van documenten voor verschillende businessdoelen gecreëerd kunnen worden. Denk bijvoorbeeld aan Gantt-planningen, *feature boards*, *wireframes*, *release plans* en strategieplannen. Daarnaast kunnen er ook op maat gemaakte rapporten gegenereerd worden op basis van de ingevoerde gegevens.

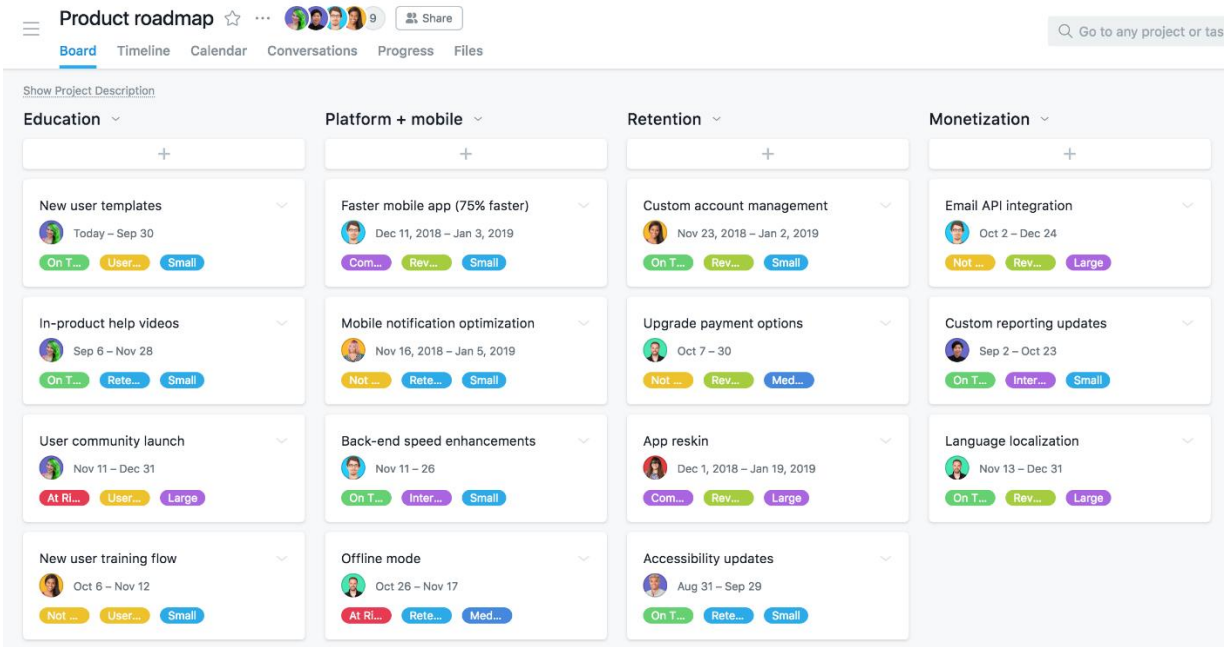



*figuur 4 - Een roadmap binnen Aha!*

De producten van Aha! zien er erg aantrekkelijk uit. Helaas voorziet Aha! de mogelijkheid niet om via Open Authorization 2 (OAuth2) in te loggen. Dit is een absolute vereiste voor Bewire, waardoor het gebruik van Aha! dus vrijwel onmogelijk is.

## Asana

Asana is een tool die courant gebruikt wordt bij tal van IT-bedrijven. Net zoals Aha! is Asana een uitgebreid softwarepakket dat voor tal van businessdoelen gebruikt kan worden. In veel bedrijven wordt Asana zelfs gebruikt als een alternatief voor Jira. Toch zijn Jira en Asana in realiteit voor andere doeleinden ontwikkeld. Jira zet zich eerder op de markt als een *software development tool* voor *agile* teams, terwijl Asana de focus eerder legt op *work tracking.* Deze nadruk komt ook erg tot uiting in de visualisatie van de roadmap in Asana. Terwijl een roadmap in één oogopslag de nodige informatie zou moeten kunnen overbrengen, wordt een roadmap binnen Asana in tal van schermen in visualisaties verdeeld. De nadruk ligt duidelijk op het traceren van het werk dat nog dient te gebeuren, voorgesteld met de nodige details zonder dat het te druk wordt. Het is echter zo dat roadmaps binnen Bewire ook gebruikt worden als communicatiemiddel naar de klanten toe, en net hier gaan de producten van Asana een beetje hun doel voorbij. Het is namelijk zo dat het gehele overzicht van de planning niet bepaald eenvoudig voorgesteld wordt, waardoor Asana dus geen gepast communicatiemiddel naar klanten toe is.




*figuur 5 - een van de schermen van een roadmap binnen Asana*

Daarnaast is Asana ook een onafhankelijk softwarepakket, waarbij out-of-the-box geen Jira-integratie mogelijk is. Om toch een intergratie tussen beiden te kunnen realiseren, is nog een bijkomende plug-in noodzakelijk. Dit betekent dus dat er, naast de kosten van Jira en Asana, ook kosten verbonden zijn aan een plug-in van een derde partij om de integratie mogelijk te maken. Hierdoor lopen de kosten meteen op. Bovendien kan er heel wat mislopen door via een bijkomende plug-in de integratie in stand te brengen, en kan een goede werking niet gegarandeerd worden. Hierdoor leek het ook twijfelachtig dat Asana het geschikte product was om het probleem van Bewire te verhelpen.

## BigPicture

BigPicture is een plug-in die vaak geprezen wordt. In tegenstelling tot Aha! en Asana, is BigPicture geen softwarepakket dat zelfstandig gebruikt kan worden. BigPicture is uitsluitend een *tool* die gebruikt kan worden als aanvulling op Jira. Naast het maken van Roadmaps, kan een gebruiker BigPicture ook gebruiken voor *resource management*, risicomanagement en Gantt-planningen. Om deze redenen is BigPicture geselecteerd voor testgebruik en om te toetsen aan de opgelegde eisen.

#### Portfolio

Portfolio is een plug-in ontwikkeld door Atlassian. Hierdoor is een zeer goede en volledige integratie verwacht. Een nadeel van Portfolio was echter dat, aangezien het een product van Atlassian is, de plug-in verbonden is aan een niet verwaarloosbaar kostenplaatje van drie euro per gebruiker per maand.

### 3.2.2.4 Onderzoeksproces

Van alle softwarepakketten die omschreven zijn in het voorgaande hoofdstuk is een selectie gemaakt van de twee meest belovende plug-ins, namelijk BigPicture en Portfolio. Deze keuze is gebaseerd op een beknopte analyse van de kwaliteit van de roadmaps, de verwachtte betrouwbaarheid en diepgang van de integratie met Jira, de gebruiksvriendelijkheid, de kostprijs en de andere mogelijke eindproducten die te ontwikkelen zijn met de software.

### 3.2.2.5 Onderzoeksresultaten

In dit onderdeel worden de belangrijkste *features* en struikelblokken van BigPicture en Portfolio omschreven, zoals ondervonden bij het uittesten van de softwarepakketten in een proefperiode van een week en op basis van de documentatie die voorzien wordt door elke producent. Voor het uitproberen van elk softwarepakket is een project aangemaakt in Jira met een aantal *epics*, waaraan *stories* gekoppeld waren. Bovendien waren er ook enkele subtaken gekoppeld aan een aantal van deze stories. Op deze manier werd een beknopt maar realistisch project nagebootst.

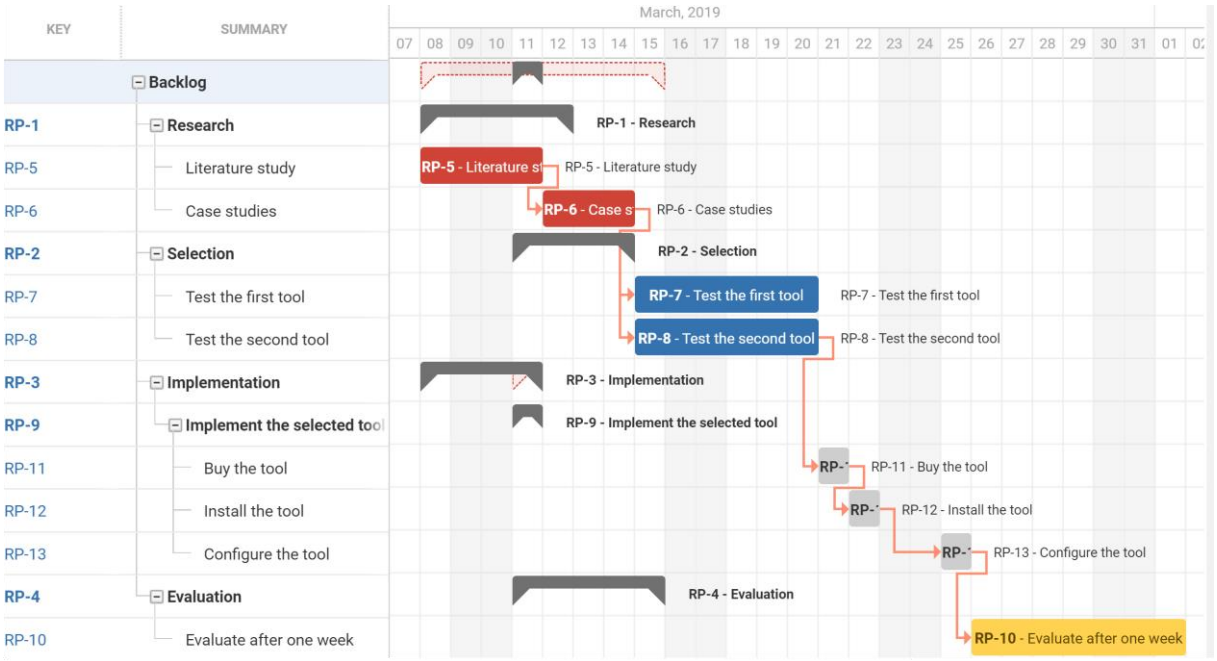
#### BigPicture

Zoals eerder omschreven kan een gebruiker naast roadmaps ook een Gantt-planning, een *resource planning* en een risicomanagementdocument opstellen. Elk van deze eindproducten werd onderzocht en zal hier omschreven worden.

Zodra BigPicture geïnstalleerd is kan de gebruiker eenvoudig één of meerdere projecten koppelen. Op deze manier kan er zelfs een overzicht over verschillende projecten heen gemaakt worden. Daarnaast kan een planning ook opgesteld worden op basis van een *Jira Query Language-query* (JQL). Op deze manier kunnen issues op een bepaalde manier gefilterd worden. Denk bijvoorbeeld aan een planning die verbonden is aan een bepaalde *developer*, een *high-level* planning waarin enkel *epics* van meerdere projecten getoond worden, enzovoort.

Wanneer BigPicture echter voor het eerst in gebruik gesteld wordt, duiken meteen al de eerste problemen op. Zo worden alle issues en al hun gekoppelde velden wel in hun geheel geïmporteerd in BigPicture, maar zijn alle onderlinge relaties wel gewist. *Stories* zijn dus niet automatisch gekoppeld aan de overeenkomstige *epic,* en zelfs de relatie tussen subtaken en hun corresponderende hoofdtaak is niet meer te bespeuren. Dit betekent dat elk van deze relaties opnieuw manueel moet

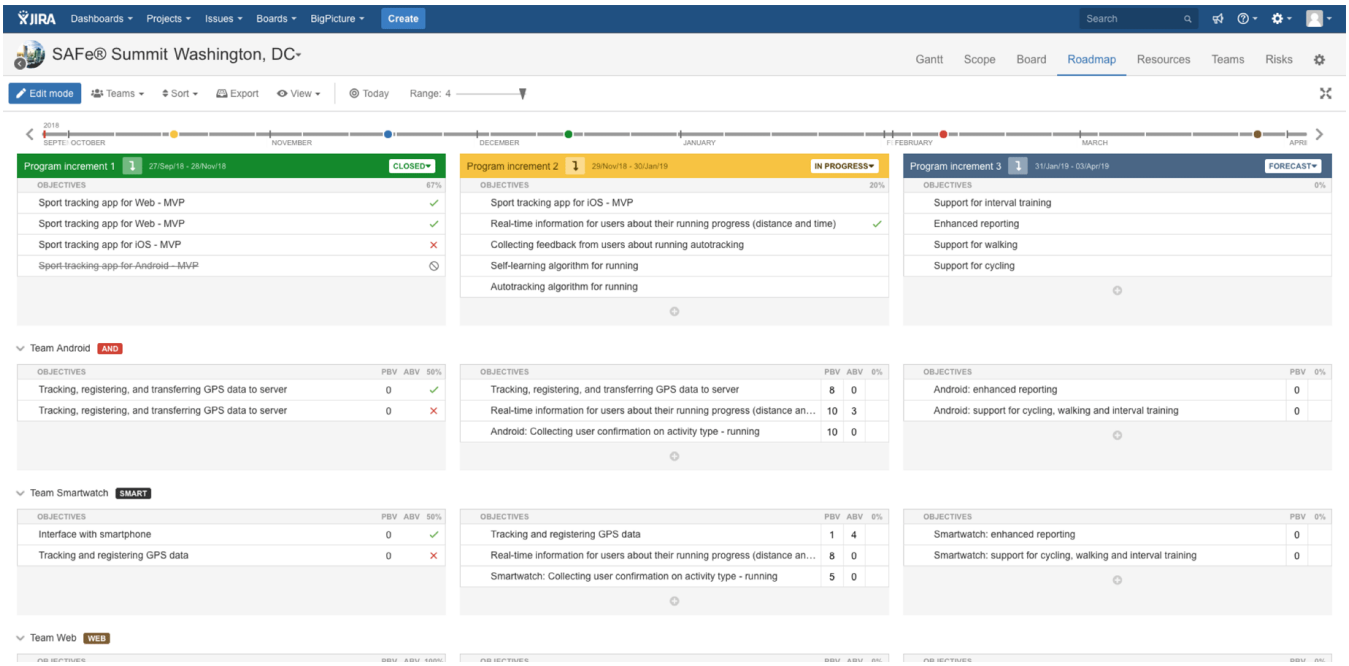
worden ingegeven wanneer het project of meerdere projecten worden ingeladen bij BigPicture. Sterker nog, deze relaties moeten aangegeven worden door het manueel verslepen van de ene *issue* onder de andere, waardoor het ook onmogelijk is om meerdere *issues* tegelijk te koppelen aan een gemeenschappelijk item. Dit is enorm tijdrovend, aangezien alle relaties al binnen Jira gelegd zijn, en dan nog eens binnen BigPicture zouden herhaald moeten worden. In complexe projecten met ingewikkelde relaties tussen tal van *issues* is dit zelfs haast onbegonnen werk.




*figuur 6 - Gantt-planning in BigPicture*

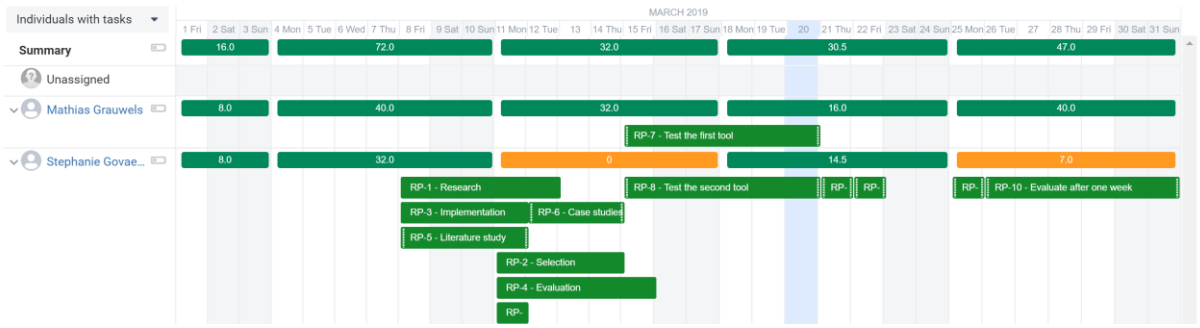
Wanneer de gebruiker er toch in geslaagd is om de relaties tussen verschillende issues opnieuw in te stellen, kan er een Gantt-planning gegenereerd worden. Helaas worden de relaties tussen gerelateerde *issues* (bijvoorbeeld "*must be done after*"), zoals aangegeven in Jira, hier ook niet gerespecteerd. Ook hier zal een gebruiker dus dubbel werk moeten leveren. Dit is niet enkel tijdrovend, maar ook gevoelig aan een verkeerde input. Bovendien worden relaties hier uitsluitend gelegd door het connecteren van de ene *issue* met de andere. Er is geen enkele manier voorzien waarop *predecessors* via tekstinput kunnen ingegeven worden, zoals gebruikelijk is in de meeste software voor projectplanning, zoals Microsoft Project of ProjectLibre. Alle relaties worden dus 'uitgetekend.' Daarnaast is het ontzettend moeilijk om de correcte visualisaties te tonen. Zoals zichtbaar in de afbeelding hierboven, komen gekoppelde *epics* niet noodzakelijk mee met hun gekoppelde issues.

Bewire had niet noodzakelijk een softwarepakket nodig om Gantt-planningen te maken. Zij gebruiken hier een ander vrijstaand softwarepakket voor. Het is immers niet nodig om de hele *backlog* op te nemen in deze planning. Het feit dat dit onderdeel niet goed werkt is niet noodzakelijk een *dealbreaker.*



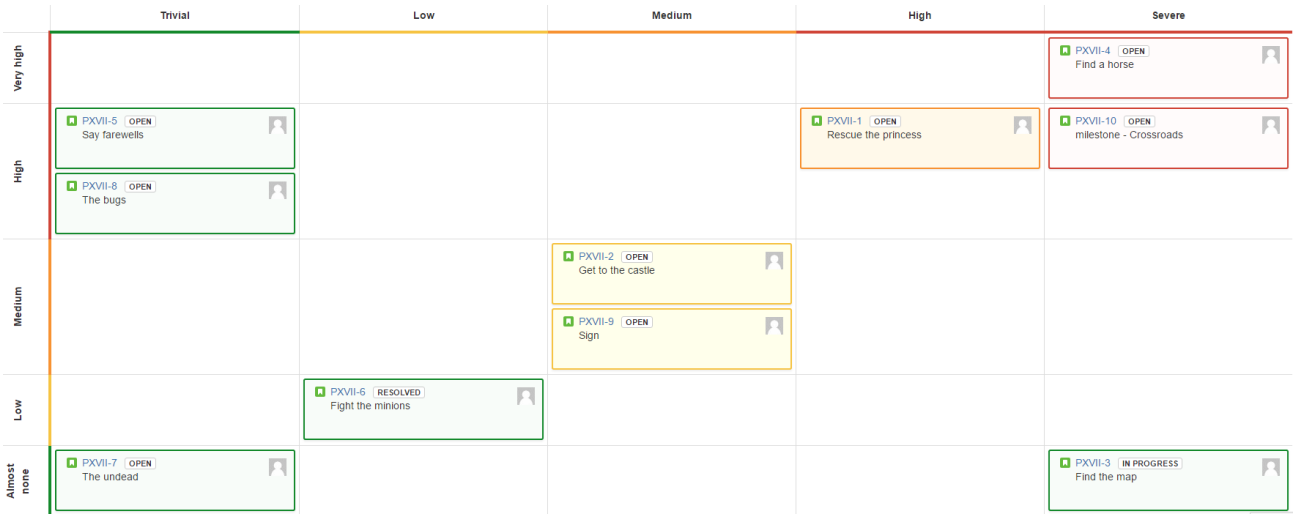

*figuur 7 - Een roadmap in BigPicture*

Wanneer er echter gekeken wordt naar de werking van roadmaps binnen BigPicture, stuiten gebruikers op dezelfde problemen. Relaties tussen gekoppelde *issues* worden niet gerespecteerd, en ook relaties tussen verschillende niveaus van *issues* (bijvoorbeeld *epic* versus *story*) worden niet in acht genomen. Dit leidt tot vreemde situaties. Stel dat een project een *epic* heeft met twee gekoppelde issues, die elk moeten opgenomen worden in de roadmap; een gebruiker zou dan verwachten dat wanneer een van de issues afgewerkt is, dat de *epic* dan ook als 50% compleet zou worden omschreven. BigPicture gaat echter elke taak afzonderlijk in rekening brengen. Dit betekent dat, door het afronden van één *issue* gekoppeld aan een *epic*, slechts 33% van het doel bereikt is. Uiteraard is deze interpretatie verkeerd, en betekent dit ook dat de roadmap van BigPicture weinig meerwaarde oplevert voor Bewire.




*figuur 8 - Resourceplanning in BigPicture*

Binnen BigPicture kunnen ook *resource plannings* opgenomen worden op basis van de *assignees* van bepaalde *issues* in Jira. Deze visualisaties zijn vrij goed en duidelijk, maar leveren onvoldoende meerwaarde op om de *tool* te installeren.

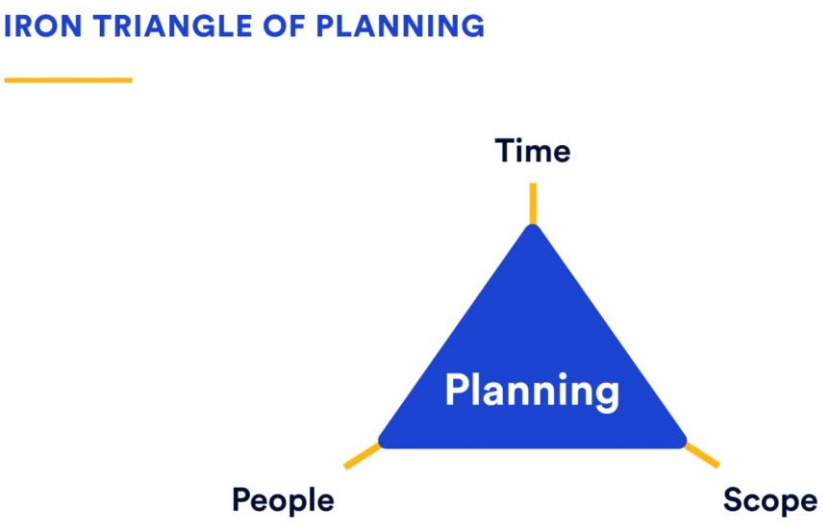



*figuur 9 - risicoplanning in BigPicture*

Tot slot kan ook een risicoanalyse opgenomen worden binnen BigPicture. Deze toepassing levert ook weinig meerwaarde op. Elk *issue* moet immers individueel op de juiste plaats binnen het schema geplaatst worden, waardoor er veel tijd verloren gaat.

## Portfolio

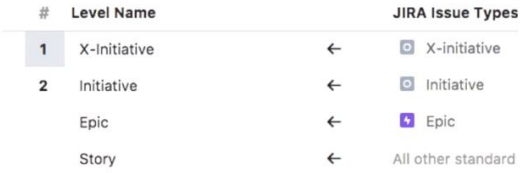
Vervolgens is er Portfolio, een plug-in voor Jira, ontworpen door Atlassian zelf. Portfolio is gebaseerd op '*the iron triangle of planning*,' oftewel de '*duivelsdriehoek*.' [3]




*figuur 10 – Duivelsdriehoek*

Voor de gebruiker aan de slag kan gaan, moet hij met een aantal aandachtspunten rekening houden. Portfolio baseert de tijdsrekeningen op de *fix versions* in Jira. Portfolio gaat immers op basis van deze *fix versions* allerlei berekeningen doen om inschattingen te kunnen maken. Denk bijvoorbeeld aan het inschatten van de uiteindelijke deadline van een project, of het inschatten van de capaciteit van

een team. Het is dus belangrijk dat *fix versions* op een consistente manier gehanteerd worden doorheen projecten. Een andere belangrijke *feature* van Portfolio is dat de gebruiker een eigen hiërarchie van *issue types* kan creëren. Simpelweg betekent dit dat de gebruiker niveaus boven een *epic* kan aanmaken, waardoor het mogelijk is om een *high-level planning* te maken. De standaard *issue types* binnen Jira zijn immers eindig, terwijl een eigen *issue type* dat niet moet zijn.

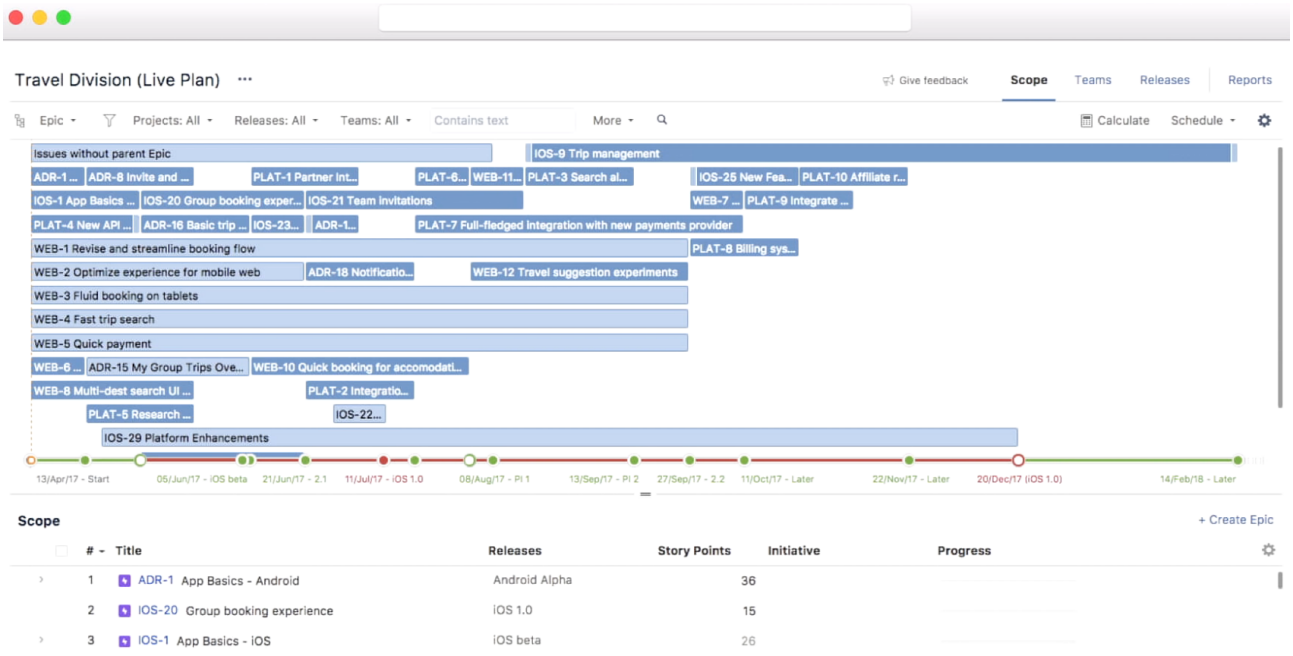



*figuur 11 - Voorbeeld van een hiërarchie van issue types in Portfolio*

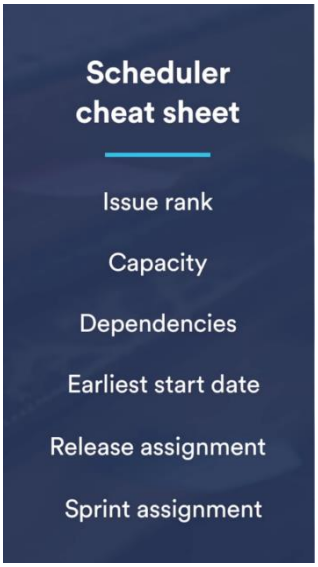
Wanneer een gebruiker aan de slag wil gaan, kan hij enerzijds Portfolio connecteren met een project in het geheel, of met één of meerdere projectborden anderzijds. Hij moet er wel rekening mee houden dat Portfolio een beperking heeft van 5000 *issues*. Om het aantal getoonde *issues* te beperken, kan er weer op basis van een JQL-query een selectie gemaakt worden van relevante *issues*. Atlassian raadt aan om steeds met een of meerdere borden te connecteren, aangezien hierdoor meer informatie beschikbaar wordt. Zo kan de *velocity* of capaciteit berekend worden op basis van sprints die al afgewerkt zijn als er van een scrum-bord vertrokken wordt. Als er geconnecteerd wordt met een kanbanbord, gebeurt er bovendien dagelijks een berekening van de capaciteit van het team, en wordt de planning dynamisch aangepast op basis van die berekeningen.

Wanneer er een connectie gemaakt is met de nodige projecten of borden, moet de gebruiker bepalen hoe hij tijd wil berekenen en weergeven. Wil hij een roadmap visualiseren op basis van maanden, of op basis van kwartalen? Misschien wil hij visualisaties op basis van releases, of misschien wil hij zelfs een meer abstracte manier om tijd voor te stellen. Zo is het bijvoorbeeld gangbaar binnen veel bedrijven om tijd uit te drukken als taken die op korte termijn, middellange termijn of op lange termijn gepland worden.

Tot slot moet de gebruiker enkel nog de nodige teams meegeven. Een zeer krachtige *feature* van Portfolio is dat de projectmanager ook rollen kan toekennen aan mogelijke teamleden. Denk bijvoorbeeld aan het onderscheid maken tussen een *front-end* en *back-end developer.* Vervolgens kan een eerste versie van de roadmap gegeneerd worden. Er zal ongetwijfeld nog wat gesleuteld moeten worden aan deze initiële planning.




*figuur 12 - Een roadmap in Portfolio*




Het is immers zo dat Portfolio alles wat ingepland kan worden, ook effectief gaat inplannen, als er geen *dependencies* aan een bepaald *issue* gekoppeld zijn. Bovendien gaat Portolio op basis van zijn eigen algoritmes prioriteiten bepalen om *issues* in te plannen. Deze volgorde is weergegeven in de figuur links.

*figuur 13 - prioriteiten van de Portfolio Scheduler*

Een ander enorm krachtige feature van Portfolio, is dat er planningen kunnen gemaakt worden op basis van *virtual contractors.* Concreet wil dit zeggen dat de projectmanager een planning kan opstellen met een virtueel team en zo tal van scenario's kan visualiseren. Zo kan hij bijvoorbeeld met een aantal mogelijke planningen naar een klant toestappen en aangeven hoe de deadline beïnvloed kan worden door extra *developer*s op het project te zetten, en mogelijke onkosten op die manier verantwoorden. In deze scenario's komt de '*iron triangle of planning*' echt tot uiting. Een klant kan immers gaan beslissen of hij het team wil vergroten, de scope wil verkleinen of de *release date* wil verplaatsen.

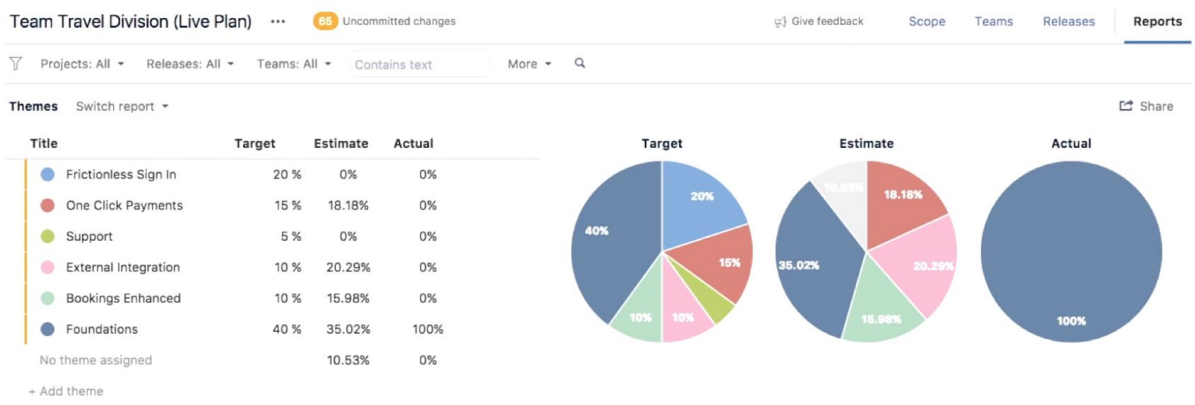

*figuur 14 - scenario's in Portfolio*

Verder geeft Portfolio ook overboekingen aan. Wanneer er onvoldoende mensen met de juiste vaardigheden op een bepaald probleem gezet worden, zal Portfolio dit visueel aangeven en zelf een haalbare deadline inschatten. Dit is opnieuw een moment waarop een projectmanager op de gepaste wijze kan ingrijpen.


*figuur 15 - visualisatie van overboeking in Portfolio*

Tot slot genereert Portfolio ook een aantal zinvolle rapporten, namelijk een *release report* en een *capacity report*. Daarnaast kan een projectmanager ook zelf focusrapporten samenstellen op basis van thema's die worden toegepast op relevante *issues*, zoals getoond in de volgende figuur. Op deze manier kunnen projecten echt gemanaged worden, en worden de gewenste situatie, de ingeschatte

situatie en de actuele situatie met elkaar vergeleken en visueel weergegeven. In één oogopslag zijn afwijkingen duidelijk.




*figuur 16 - Focus report op basis van thema's*

### 3.2.2.6 Besluit

BigPicture is in realiteit zo ongebruiksvriendelijk en beperkt in functionaliteit, dat de software absoluut geen meerwaarde voor Bewire zou zijn in gebruik. Portfolio daarentegen vinkt vrijwel alle eisen af, zoals zichtbaar is in de matrix hieronder.


*figuur 17 - MoSCoW-matrix toegepast op Portfolio*

## 3.2.3 User groups & project roles in Jira

### 3.2.3.1 Inleiding

Binnen Jira kan er gewerkt worden met *user groups* en *project roles*. Gebruikers kunnen zo toegekend worden aan een *user group* of kunnen een bepaalde projectrol toegekend worden. Op

deze manier kan de Jira-administator bepaalde rechten toekennen aan gebruikers, hetzij op globaal niveau of echt op projectniveau. Daarnaast kan de administrator dankzij groepen en rollen ook configureren welke gebruikers bepaalde e-mailnotificaties krijgen, welke *workflow conditions* toegekend kunnen worden aan bepaalde gebruikers, enzovoort.

#### 3.2.3.2 Probleemstelling

Wanneer nieuwe gebruikers aan een Jira-instantie toegevoegd worden, worden zij automatisch toegekend aan de *Jira-software-users-group.* Nieuwe gebruikers krijgen bovendien onmiddellijk toegang tot alle bestaande projecten en dashboards op die *instance*. Op deze manier kunnen zij meteen aan de slag in bestaande projecten, en moeten jira-administrators zich geen zorgen maken om eerst de gepaste rechten te configureren. Toch is deze functionaliteit niet altijd wenselijk. In sommige gevallen moeten nieuwe gebruikers beperkte rechten krijgen, waardoor zij niet meteen toegang hebben tot lopende projecten. Denk bijvoorbeeld aan stagiairs of externen die mogelijk wel toegang moeten krijgen tot een eigen project, maar geen inzage mogen hebben in andere projecten uit veiligheidsoverwegingen of omwille van gevoelige informatie.

#### 3.2.3.3 Uitwerking

Voor de praktische uitwerking om dit probleem aan te pakken, is er een handleiding geschreven. Deze uitgewerkte handleiding is terug te vinden als bijlage.

Een belangrijke opmerking is dat om deze handleiding te kunnen volgen, de gebruiker de allerhoogste rechten nodig heeft, namelijk de rechten van *site-administrator.*

### 3.2.4 GDPR-compliancy binnen Jira

#### 3.2.4.1 Inleiding

De *General Data Protection Regulation (GDPR)*, oftewel de Algemene Verordening Gegegevensbescherming (AVG), is een nieuwe Europese wetgeving die in voegen is gegaan op 25 mei 2018. GDPR gaat over het beheer en beveiliging van persoonlijke gegevens van Europeanen. Concreet betekent deze wetgeving dat organisaties zich vanaf heden moeten kunnen verantwoorden voor de persoonsgegevens die ze verzamelen, hoe ze deze gaan gebruiken en hoe deze data beveiligd wordt. [4]

#### 3.2.4.2 Probleemstelling

Persoonsgegevens worden op verscheidene manieren gebruikt binnen Bewire. Bewire heeft een eigen *Data Protection Officer* (DPO) aangesteld, om te garanderen dat de GDPR-regelgeving correct wordt nageleefd. Deze functie is binnen het bedrijf weggelegd voor Jens Mortier.

Toch was het zinvol om nog even te onderzoeken hoe de GDPR-wetgeving zo goed mogelijk kan nageleefd worden binnen Jira. Het is immers zo dat Atlassian enige tijd heeft nodig gehad om zichzelf als *GDPR-compliant* te kunnen bestempelen. Zo is het dat velden met gevoelige informatie binnen Jira, bijvoorbeeld gebruikersnamen, pas voldeden aan de GDPR-wetgeving op 29 maart 2019. Met

andere woorden, Atlassian heeft enkele aanpassingen gedaan aan de werking van de software in de loop van de stageperiode. Vandaar dat het zeker een meerwaarde was om even na te gaan wat er precies veranderd was, en welke maatregelen in het algemeen beter gevolgd werden.

#### 3.2.4.3 GDPR-compliancy door Atlassian

*GDPR-compliancy* moet eigenlijk gerealiseerd worden door zowel de gebruiker als door de *provider* van een dienst. Dit betekent dat enerzijds Atlassian de nodige stappen zal moeten ondernemen om zelf *GDPR-compliant* te zijn, en anderzijds een gebruiker zich aan *compliance*-regelementen zal moeten houden.

Atlassian heeft zelf een aantal maatregelen getroffen om aan de opgelegde wetgevingen te voldoen. Zo hebben ze noemenswaardige *upgrades* gedaan aan hun infrastructuur en hebben ze ondertussen een aantal *security*-certificaten behaald, zoals ISO/IEC 27001, ISO/IEC 27002, en ISO/IEC 27018 voor Jira Software, Jira Service Desk, Jira Core en Confluence Cloud. [5] Andere producten van Atlassian zouden eveneens zo snel mogelijk gecertificeerd worden.

Verder voorziet Atlassian ook een *framework* om data buiten Europese grenzen te transfereren volgens GDPR-normen. [5]

De GDPR-wetgeving eist ook dat organisaties voldoen aan *'Data Portability.'* Dit betekent dat alle data die een organisatie van een persoon bewaart, mogelijk opleverbaar moet zijn aan die persoon. Hiervoor voorziet Atlassian de mogelijkheid om alle data te exporteren.

Tot slot is er ook '*The Right to be Forgotten*.' Hiermee wordt binnen GDPR geëist dat alle persoonlijke gegevens op aanvraag in bepaalde omstandigheden onmiddellijk gewist moeten kunnen worden uit alle *databases* en andere bronnen van een organisatie. Om aan deze regelgeving tegemoet te komen, voorziet Atlassian de mogelijkheid om gebruikers eenvoudig uit systemen te wissen enerzijds. Anderzijds wordt er ook rekening gehouden met mensen die niet rechtstreeks gebruiker zijn van de producten van Atlassian, maar waarvan wel persoonlijke gegevens bewaard worden, zoals klanten van gebruikers. Zij kunnen namelijk ook een aanvraag om gewist te worden, indienen.

#### 3.2.4.4 GDPR-compliancy door de gebruiker

Wanneer bedrijven gebruik maken van producten van Atlassian, moeten zij zich ook houden aan de GDPR-wetgeving. Dit betekent dat zij zich erg bewust moeten zijn van de data die zij van klanten of werknemers bijhouden binnen bijvoorbeeld Jira of Confluence.

Bedrijven moeten in ieder geval duidelijk aangeven aan klanten welke persoonlijke gegevens bewaard worden, voor welke doeleinden deze gegevens gebruikt gaan worden, en hoe lang deze gegevens bewaard worden.

Bovendien moet elk bedrijf ook kunnen voldoen aan *'The Right to be Forgotten'* en *'Data Portability,'* moest dit van toepassing zijn. Alle gegevens van klanten of werknemers moeten dus op aanvraag opgeleverd kunnen worden, of in hun geheel gewist worden.

Deze regelgeving is dus ook geldig binnen alle producten van Atlassian die in gebruik zijn. Hiervoor zijn een aantal aandachtspunten en mogelijke zinvolle werkwijzen opgesomd.

### Persoonlijke gegevens beperken

Een eerste voor de hand liggende werkwijze is dat een organisatie gewoonweg de hoeveelheid persoonlijke gegevens die ze bewaart, verkleint. Aanvankelijk lijkt het misschien een goed idee om zoveel mogelijk gegevens van klanten te bewaren, ongeacht of het bedrijf nu een zinvol gebruik van deze gegevens voor ogen heeft. Toch is dit geen goed idee. Enerzijds mogen gegevens volgens GDPR enkel bewaard worden als het bedrijf het verzamelen en het gebruik ervan kan verantwoorden en kan meedelen aan de klanten. Anderzijds kan het ook gewoon moeilijker zijn om gegevens op een correcte, veilige en overzichtelijke manier te bewaren, als er ontzettend veel gegevens bewaard worden. Het is dus een goede reflex om goed na te denken welke gegevens absoluut noodzakelijk zijn om de businessdoelen te kunnen bereiken, hoe het verzamelen en gebruik hiervan verantwoord kan worden, en dan ook uitsluitend die gegevens te verzamelen en bewaren, zowel binnen producten van Atlassian als daarbuiten.

Daarnaast is het soms ook mogelijk om de gegevens op een andere, niet persoonsgebonden manier op te slaan. Wanneer er in Jira bijvoorbeeld *features* worden geschreven voor een specifieke klant, kan er naar deze persoon verwijzen worden door zijn rol binnen het bedrijf te benoemen, eerder dan zijn naam. Gebruik dus niet ‘Jan Peeters’, maar wel ‘de officemanager van bedrijf X’.

### Actief verwijderen van oude data

Organisaties moeten ook steeds aan hun klanten aangeven hoe lang ze bepaalde persoonsgegevens gaan bijhouden. Het is dan ook de bedoeling dat deze gegevens na de beloofde termijn actief verwijderd worden. Terwijl gegevens in een ideale wereld centraal bewaard worden, is dit in de praktijk vaak onmogelijk.

Ook binnen de producten van Atlassian kan het gebruik van persoonlijke gegevens soms noodzakelijk zijn. Zo worden klanten soms bij naam genoemd in een *issue*, worden e-mailadressen om praktische redenen toegevoegd of worden er bijlages met gevoelige informatie toegevoegd aan een ticket, enzovoort.

Het kan soms een bijna onmogelijke taak zijn om deze gegevens na maanden of zelfs jaren op te sporen en te verwijderen. Vandaar is het belangrijk om hiervoor een vaste werkwijze te bepalen en deze vervolgens consistent blijven toepassen.

Allereerst moet er nagegaan worden hoe persoonsgegevens eenvoudiger te traceren zijn binnen Jira. Een mogelijke werkwijze is een *keyword* toevoegen aan de omschrijving van een *issue* in Jira, of zelfs een GDPR-label toevoegen aan elk *issue* dat persoonlijke informatie bevat. Op deze manier kunnen *issues* in de toekomst via een *JQL-query* gefilterd worden op basis van dit *keyword* of label. Bovendien kunnen hier ook weer automatisatieregels worden toegepast, zoals bijvoorbeeld het automatisch verwijderen van bijlages van *issues* die het *keyword* bevatten na een bepaalde termijn.

### Minimisation

Naast het zoveel mogelijk beperken van de persoonlijke informatie die verzameld en bewaard wordt, is er ook het concept van *minimisation* binnen GDPR. Met *minimisation* wordt bedoeld dat persoonlijke gegevens centraal bewaard moeten worden, en dat deze gegevens elders zo min mogelijk gereproduceerd mogen worden. Zo is het goed om een centrale database of een *Customer*

*Relationship Management Tool (CRM-tool)* te hebben waarin de gegevens bewaard worden, zodat fragmentatie van de gegevens voorkomen kan worden.

#### Anonymisation

Verder is het ook gebruikelijk om persoonlijke gegevens te gaan anonimiseren. Zo kan een gebruikersnaam binnen tal van softwareprogramma's tegenwoordig vervangen worden door een identificatienummer (ID). Ook binnen Jira is het zo dat vanaf 29 april 2019 geen *usernames* of *user keys* gebruikt konden worden. Jira heeft hiervoor ook het gebruik van identificatienummers mogelijk gemaakt om de *usernames* te vervangen. Dit betekende echter wel dat deze verandering in de loop van de stage doorgevoerd werd.

Een belangrijke opmerking is ook dat een ID niet in elk scenario kan gebruikt worden als alternatief voor een persoonlijk gegeven zoals een *username*. Een ID kan uiteindelijk ook teruggekoppeld worden naar de eigenaar, waardoor een ID in bepaalde gevallen ook als een persoonsgegeven beschouwd kan worden.

#### Aandacht schenken aan werknemersgegevens

Een aspect van GDPR dat vaak over het hoofd gezien wordt, is het feit dat persoonlijke gegevens van werknemers ook onderhevig zijn aan de wetgeving. Een bedrijf moet dus niet uitsluitend rekening houden met gegevens van klanten, maar ook met persoonlijke informatie van werknemers of onderaannemers.

Binnen Jira en andere software van Atlassian moet er ook absoluut rekening gehouden worden met dit aspect. Een werknemer die bijvoorbeeld zijn ontslag ingediend heeft, wil mogelijk ook *zijn 'Right to be Forgotten'* uitoefenen. In dat geval moet de Jira-administrator deze persoon volledig uit Jira verwijderen. Zoals eerder aangegeven voorziet Jira de functie om gebruikers uit het systeem te verwijderen, en daarmee ook alle informatie van deze gebruiker uit de corresponderende programma's te verwijderen. Toch moet een administrator voorzichtig met deze functionaliteit omgaan. Wanneer een gebruiker onmiddellijk gewist zou worden, kan dit enorme problemen in Jira veroorzaken, wanneer deze gebruiker gekoppeld is aan bepaalde *issues*. Deze gebruiker moet dus eerst binnen de Atlassian suite gemarkeerd worden als inactief, vervolgens moeten zijn gegevens gewist of geanonimiseerd worden. Pas na deze stappen mag de gebruiker uit het systeem gewist worden.

### 3.2.5 Synchronisatie tussen projecten en Jira-instanties

#### 3.2.5.1 Inleiding

Tijdens het onderzoek naar *GDPR-compliancy* binnen de Atlassian suite, is *Exalate* een tool die geprezen werd in het kader van GDPR binnen Jira. Exalate is een plug-in waarmee verschillende Jira-projecten of zelfs verschillende Jira-instanties met elkaar gesynchroniseerd kunnen worden. Deze plug-in zou mogelijk een enorme meerwaarde voor Bewire kunnen creëren. Na overleg met de bedrijfspromotor zijn dan ook de mogelijkheden onderzocht.

#### 3.2.5.2 Probleemstelling

Bewire werkt, naast aan interne projecten, ook aan externe projecten op locatie, op basis van consultancy. Dit betekent dat Bewire-consultants vaak op de *Jira instances* van klanten terechtkomen. *Jira workflows* kunnen van bedrijf tot bedrijf enorm verschillen. In de praktijk betekent dit dat er heel wat tijd geschonken moet worden aan het begrijpen en kunnen volgen van bestaande externe *workflows*. Het is ook mogelijk dat Bewire-projectmanagers deze *workflows* nog moeten aanpassen of zelfs een nieuwe *workflow* moeten opzetten. Dit is een tijdsrovend proces waardoor hier vaak ook heel wat kosten voor de klanten aan verbonden zijn. Een *tool* zoals Exalate zou ervoor kunnen zorgen dat Bewire-consultants externe *issues* gewoon kunnen synchroniseren met een Bewire-Jira-instantie, en met de interne *workflow* aan de slag kunnen op externe projecten. Bovendien betekent dit dat ook de automatisatie en *pipeline*, zoals eerder beschreven, ook behouden kan blijven, waardoor alle partijen eigenlijk geholpen zijn.

Een andere probleemstelling is bijvoorbeeld dat er verschillende teams zijn die op één Jira-instantie aan een gezamenlijk doel willen werken, maar daarvoor nu noodgedwongen in één project moeten werken, ongeacht hun rol of aandeel in het project, en dus onderworpen worden aan de gehele *backlog* en de gehele *workflow*.

#### 3.2.5.3 Voorbeeld case: Development & QA

##### Situatieschets

Bewire heeft momenteel geen intern QA-team. Voor het *development process* van bepaalde applicaties is een QA-team mogelijk wel noodzakelijk. Bewire kan er dus voor kiezen om een extern QA-team in te schakelen. Bewire kan vervolgens verkiezen dat het QA-team op in interne *Bewire-Jira-instance* aan de slag gaat, weliswaar met beperkte toegang en afgeschermd van andere projecten.

##### Setup

In dit scenario wordt ervan uitgegaan dat het QA-team op een interne Jira-instantie van Bewire zou werken. Met Exalate is het ook mogelijk om twee verschillende *instances* met elkaar te synchroniseren, maar dit scenario is niet binnen deze stageperiode uitgevoerd geweest. Bovendien is de configuratie van Exalate met twee interne projecten, of twee projecten met een interne en externe *instance* gelijkaardig. Het zou dus geen meerwaarde zijn dit scenario uit te testen en daarbij ook nog een bijkomende licentie aan te kopen.

Een ander aandachtspunt is dat in dit voorbeeld beide teams een andere Jira-*workflow* hanteren. Dit is realistisch, aangezien beide teams een andere werking en andere doelstellingen hebben.

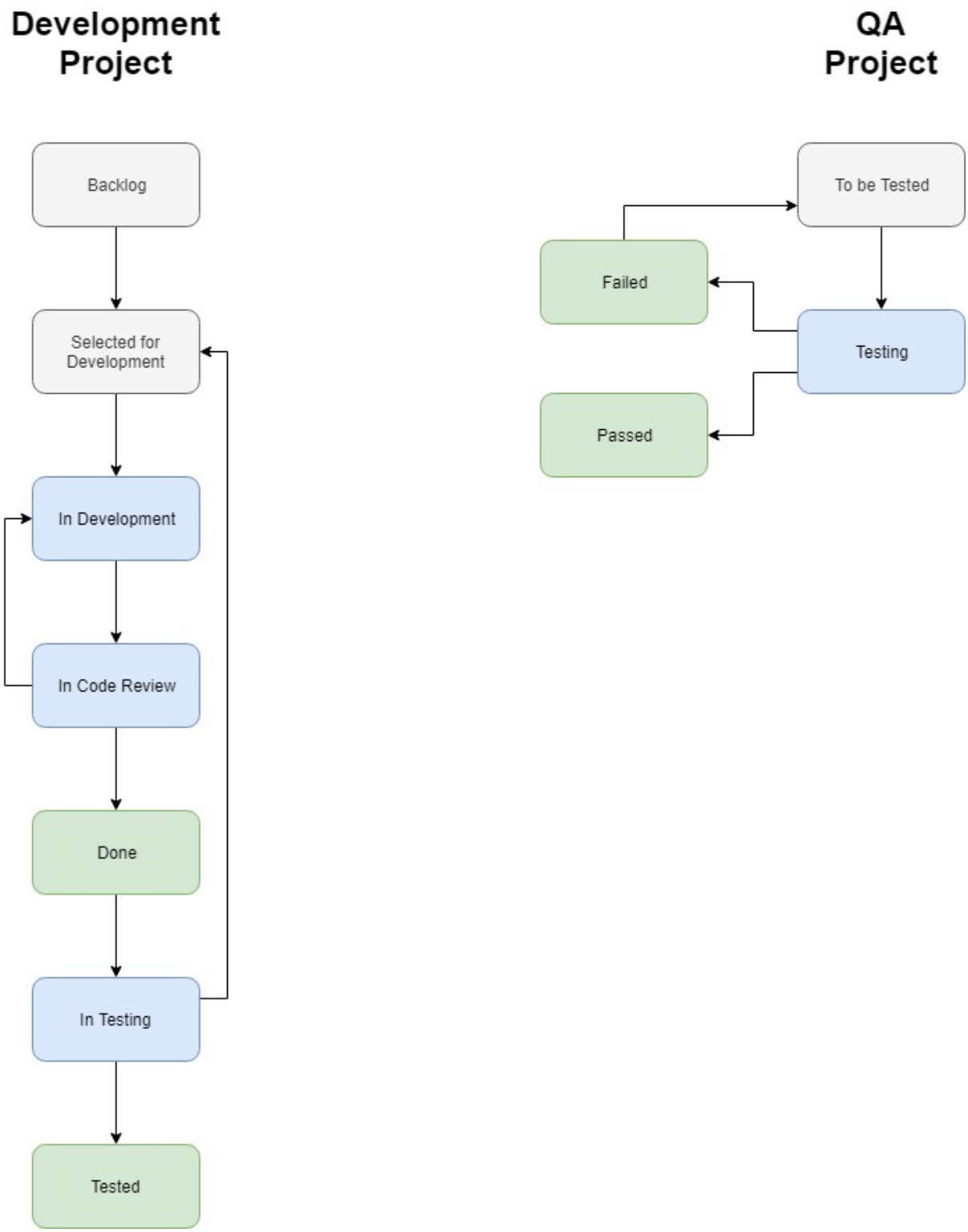
Verder is het ook de bedoeling dat het QA-team bepaalde issues enkel moet opvolgen wanneer zij hiermee aan de slag moeten. Concreet betekent dit dat een ticket pas naar het QA-kanbanbord gestuurd mag sturen, wanneer dit ticket klaar is om getest te worden.

Daarnaast is het ook belangrijk dat er gewerkt is met bi-directionele synchronisatie. Issues moeten naar QA gestuurd worden wanneer ze klaar zijn om getest te worden, maar QA moet ook issues van het type bug kunnen aanmaken, en vervolgens naar de *developers* doorspelen.

Tot slot is het ook de bedoeling dat issues automatisch naar het gewenste project gestuurd worden, en dat hier geen manuele handeling nodig is.

### Workflows

Op de volgende afbeelding staan de *workflows* van elk team. De ontwikkelaars gebruiken de *workflow* zoals ook in het onderzoekstopic omschreven staan. Deze workflow is zwaar geautomatiseerd. Daarentegen gebruikt het QA-team een eerder beperkte workflow, waarin zij uitsluitend een *backlog* hebben van *issues* die getest moeten worden, vervolgens een status waarin zij *issues* kunnen plaatsen eens ze getest worden en tot slot statussen voor *issues* waarvan de testen slagen of falen.




*figuur 18 - workflows van QA en Development*

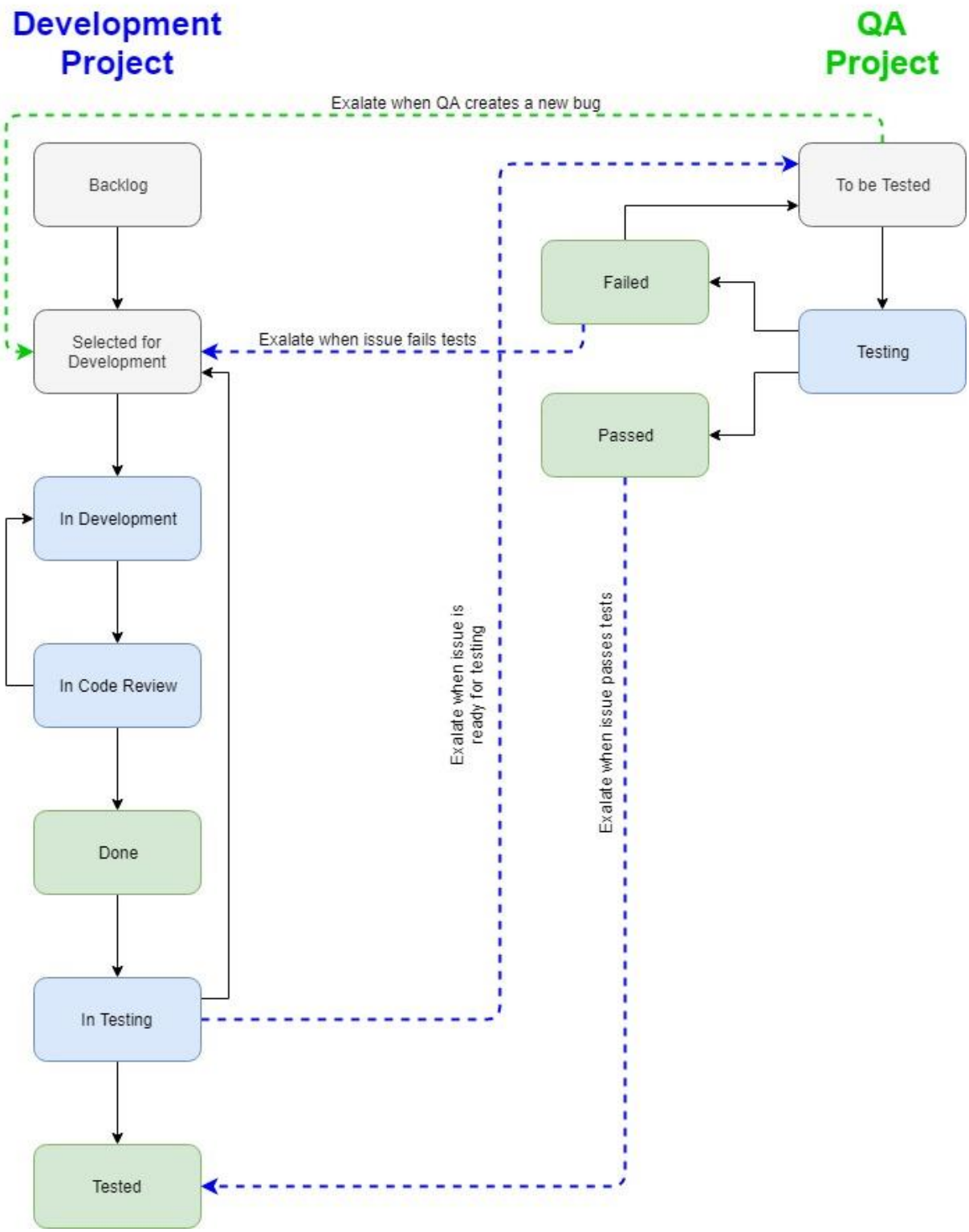
### Doelstellingen

Beide teams werken in hun eigen project op hun eigen kanbanbord. Wanneer de ontwikkelaars aan de slag gaan met het project, is het de bedoeling dat ze hun flow (afgeschermd van het QA-team) kunnen doorlopen, tot wanneer een issue de status '*Done'* bereikt. Vervolgens is het de bedoeling dat *developer*s manueel de issues gaan verplaatsen naar "In Testing." Op dit moment zou het ticket via Exalate moeten verschijnen in '*To be Tested*,' waardoor de testers vervolgens aan de slag kunnen gaan met het testen, door het *issue* manueel te verplaatsen naar '*Testing.*'

Wanneer de testen afgerond zijn, kan het QA-team aangeven of de testen al dan niet geslaagd zijn, door het overeenkomstige ticket naar '*Failed*' of '*Passed*' te verplaatsen. Een *issue* waarvan de testen geslaagd zijn, moet vervolgens bij het *development-team* automatisch verplaatst worden naar '*Tested,'* wat ook de finale status is in hun workflow. Issues waarvan de testen echter niet slagen, moeten bij QA verplaatst worden naar '*Failed.*' Deze handeling zorgt ervoor dat het ticket bij het *development team* automatisch van '*in testing*' terug naar '*Selected for Development*' wordt verplaatst.

Tot slot is het ook de bedoeling dat het QA-team op eigen initiatief *bugs* kan creëren en deze kan linken aan bestaande *issues* die zij ontvangen. Deze *bugs* moeten op hun beurt dan weer naar het *development team* gestuurd worden en daar in '*Selected for Development*' geplaatst worden.

Op de volgende afbeelding zijn deze transities met een doorbroken lijn aangegeven. *Issues* die ontstaan in het *development project,* zijn aangegeven met een blauwe lijn, terwijl *issues* die bij QA ontstaan, zijn aangegeven in het groen.

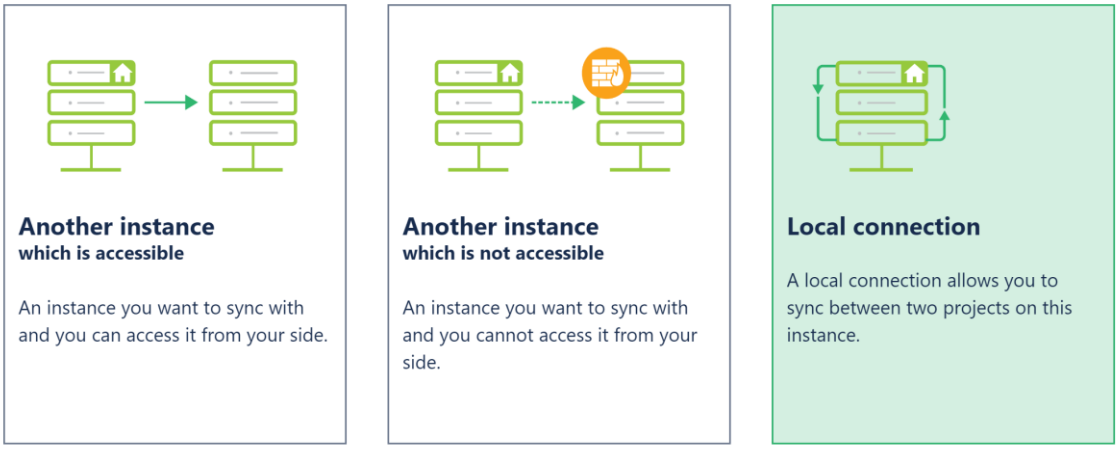



*figuur 19 - Automatische Exalate transities*

### Werkwijze

Uiteraard moet een projectmanager beide projecten eerst aanmaken, de *workflows* creëren en vervolgens toekennen aan het gewenste project. Vervolgens kan de configuratie van Exalate gebeuren.

Exalate kan, zoals eerder aangegeven, projecten op een lokale connectie met elkaar synchroniseren, zoals het geval is in dit voorbeeld. Daarnaast kan Exalate ook projecten op verschillende *Jira instances* met elkaar synchroniseren. Initieel zal Exalate ook vragen naar het type connectie dat de gebruiker wil realiseren. Het configuratieproces voor een lokale connectie verschilt enigszins van het connecteren met een externe *instance.* Bovendien heeft een Bewire-projectmanager mogelijk niet voldoende rechten om zelfstandig de configuratie op de externe instantie te vervolledigen. Mogelijk moet er dus ook wat afgestemd worden met de administrator van de externe partij.




*figuur 20 - mogelijke connectietypes van Exalate*

Eens de connectie gemaakt is, is verdere configuratie noodzakelijk. Eerst moeten *sync rules* geschreven worden. In deze regels wordt aangegeven welke eigenschappen van een issue gesynchroniseerd worden. Om *GDPR-compliant* te zijn kan er bijvoorbeeld gekozen worden om bepaalde velden niet mee te synchroniseren. Zo kan Bewire bijvoorbeeld kiezen om geen bijlages mee te sturen naar het QA-team, of om bijvoorbeeld niet kenbaar te maken wie de interne *assignee* van een *issue* is.

Daarnaast moet ook aangegeven worden welke statussen van het ene project overeenstemmen met statussen uit het andere. Out of the box kan Exalate immers uitsluitend *issues* tonen in een ander project, zonder dat statussen mee gesynchroniseerd worden.

Elk van bovengenoemde instellingen, moeten gebeuren aan de hand van *Groovy scripts* en de *Application program interface (API)* van Jira. Het is dus noodzakelijk dat de persoon die deze configuratie moet uitvoeren enige technische bagage heeft. Deze scripts moeten immers voor elk project, waarbij aan minstens één zijde veranderingen zijn gebeurd aan de *workflow*, opnieuw geschreven worden, dus er is zelden de mogelijkheid om de scripts simpelweg te kopiëren voor een nieuwe connectie. Bovendien is er ook een verschil in code tussen lokale connecties enerzijds, en connecties tussen verschillende *instances* anderzijds. Wanneer twee verschillende instanties met elkaar geconnecteerd worden, gebeurt het synchroniseren tussen statussen op codeniveau aan de

hand van een *mapper.* Bij het connecteren van lokale projecten kan dit gebeuren via een eenvoudig *if-statement*.

Voor het scenario dat we hier omschrijven krijgen we dan bijvoorbeeld het volgende script:

Incoming sync for existing issues (change processor)

```
1  if(replica.project.key == "DEV" && replica.status.name == "In Testing"){
2      workflowHelper.transition(issue, "To be Tested")
3  }
4  if(replica.project.key == "QA" && replica.status.name == "Passed"){
5      workflowHelper.transition(issue, "Tested")
6  }
7  if(replica.project.key == "QA" && replica.status.name == "Failed"){
8      //issue.typeName = "Bug"
9      workflowHelper.transition(issue, "Selected for Development")
10 }
11 if(replica.project.key == "QA" && replica.status.name == "To be Tested"){
12     workflowHelper.transition(issue, "Selected for Development")
13 }
14 issue.summary      = replica.summary
15 issue.description  = replica.description
16 issue.labels       = replica.labels
17 issue.comments     = commentHelper.mergeComments(issue, replica)
```

*figuur 21 - een van de scripts die geschreven moeten worden voor Exalate*

Door deze scripts te schrijven, bereikt de projectmanager nog steeds niet de gewenste functionaliteit. De statussen zijn wel gesynchroniseerd, maar de *issues* worden nog niet automatisch overgeheveld van het ene project naar het andere. Concreet betekent dit dat een *issue* dat bijvoorbeeld bij de ontwikkelaars in '*In testing*' staat, enkel via een manuele klik op de Exalate-knop naar '*To be tested*' bij QA verplaatst kan worden. Uiteraard is het de bedoeling dat dit automatisch gebeurt op basis van transities van *issues* binnen één project.

Deze automatisatie is op verschillende manieren te verkrijgen, zoals hieronder opgesomd:

*a)* *Issues* in bulk synchroniseren op basis van een *JQL-query*
b) Synchronisatie via de *post-function* op *workflow*-niveau
c) Het schrijven van Exalate-triggers

Het schrijven van Exalate-triggers is in dit scenario veruit de meest gebruiksvriendelijke en overzichtelijke manier om synchronisaties in gang te zetten. Alle automatisatieregels staan dan immers op één locatie, waardoor het overzicht eenvoudig bewaakt kan worden. Een dergelijke trigger wordt aan de hand van een JQL-filter geschreven, zoals zichtbaar is in de volgende figuur.


*figuur 22 - Exalate triggers*

De triggers die hierboven staan afgebeeld, gaan telkens een *issue* aangemaakt of geüpdatet wordt in één van de gekoppelde projecten, elke JQL-query uitvoeren. Wanneer een *query* één of meerdere *issues* teruggeeft, worden deze *issues* in het gekoppelde project gesynchroniseerd. Op die manier wordt de statussynchronisatie, zoals aangegeven in de Groovy-scripts, gerealiseerd.

### Aandachtspunten

Exalate kan een ontzettend krachtig middel zijn voor bedrijven die werken met onderaannemers of op basis van consultancy op locatie bij klanten aan de slag gaan. Ook voor Bewire kan Exalate dus een enorme meerwaarde opleveren. Toch zijn er een paar aandachtspunten waarmee rekening gehouden worden vooraleer de keuze om met Exalate aan de slag te gaan, gemaakt kan worden, namelijk:

a) Om Exalate te configureren is enige technische kennis noodzakelijk.
b) Wanneer er tussen twee instanties gesynchroniseerd wordt, moet de volledige setup en configuratie op zowel de interne, als de externe instantie gebeuren.
c) Wanneer er synchronisatie tussen meerdere instanties opgezet wordt, moet er uiteraard een licentie voor elke instantie aangekocht worden.
d) Scripts moeten voor elk individueel project geschreven worden.
e) Niet elk veld van een *issue* moet gesynchroniseerd worden. Hierdoor kan Exalate ook een middel zijn om velden met gevoelige data niet mee te synchroniseren naar derden. Dit is enorm waardevol in het kader van GDPR.

## 3.3 Security en quality

Binnen dit onderzoeksdomein zijn security en kwaliteit bestudeerd. Alle onderzoeksresultaten en implementaties die betrekking hebben tot security, zijn opgenomen in het eindwerk van mijn collega, Mathias Grauwels. Binnen dit onderdeel worden de concrete onderzoeksresultaten naar tal van *tools* met betrekking tot kwaliteit en eventuele implementaties omschreven.

### 3.3.1 Code Quality

#### 3.3.1.1 Inleiding

Vooraleer er concrete stappen ondernomen kunnen worden om de *code quality* te meten, controleren en verbeteren, moet er een besef zijn van wat *code quality* juist is. *Code quality* kan immers op talloze manieren geïntepreteerd worden, en is bovendien afhankelijk van de sector waarin er ontwikkeld wordt. Zo zal een *automotive developer code quality* helemaal anders interpreteren dan een web *developer.* [6]

*Code quality* is bovendien cruciaal voor de algehele softwarekwaliteit, vooral in veiligheidsgevoelige systemen.

Bovendien heeft onderzoek aangetoond dat het ontzettend moeilijk is voor ontwikkelaars om defecten in hun eigen code te identificeren. Zo zijn ontwikkelaars maar voor minder dan 50% efficiënt in het vinden van *bugs.* [7]

Er is geen eensgezinde definitie van wat kwaliteitsvolle code is. Zo kan enerzijds gesteld worden dat kwaliteitsvolle code eenvoudig is, *bug free* is, getest is, onderhoudbaar is, *gerefactord* is, goed gedocumenteerd is, enzovoort. Het is dus zeer belangrijk dat er duidelijk naar het team gecommuniceerd wordt welke vereisten er binnen de organisatie of binnen een project zijn om code als kwaliteitsvol te kunnen bestempelen.

#### 3.3.1.2 Complexiteit van code meten

Bepaalde applicaties steunen tegenwoordig op complexe code. Hoe complexer code echter is, hoe moeilijker het is de code te onderhouden en hoe onbetrouwbaarder de code is [8]. Een van de aspecten die *code quality tools* dan ook bestuderen, is de complexiteit van de code. Zo zullen complexe codeblokken of methodes door de *code quality tools* herkend worden, waardoor *developer*s vervolgens deze code kunnen herschrijven, of uitvoeriger kunnen testen. Bovendien kunnen mogelijke risico's hierdoor geïdentificeerd worden, is de huidige status van een project begrepen en kan de voortgang van een project gerichter opgevolgd worden. [8]

Er zijn heel wat *metrics* waarmee rekening gehouden wordt, en verschillende *code quality tools* gebruiken doorgaans een verschillende combinatie van *metrics* en algoritmes om de complexiteit van code te bespreken. Hieronder staan de belangrijkste *metrics*, die doorgaans in elke *code quality tool* gebruikt worden.

Enkele *metrics* of algoritmes die doorgaans gebruikt worden binnen de meeste *code quality tools* zijn *cyclomatic complexity, ABC software metric* en simpelweg de *lines of code.* [9]

#### 3.3.1.3 Technical debt

*Technical debt* of technische schuld is, is de term die gebruikt wordt wanneer niet-optimale code wordt opgeleverd. Deze code kan onduidelijk geschreven zijn, of nog elders *bugs* veroorzaken, en zou dus op een later moment verbeterd moeten worden. [9]

Vaak wordt technische schuld veroorzaakt wanneer een bepaalde *feature* onmiddellijk uitgebracht moet worden, door bijvoorbeeld businesseisen.

Daarnaast wordt technische schuld ook veroorzaakt door code die pas op termijn problemen zal veroorzaken, op te leveren. Een bekend voorbeeld hiervan is de millenniumbug. [10]

Technische schuld zorgt ervoor dat projecten steeds moeilijker worden om te onderhouden. Hoe meer code geschreven is, hoe waarschijnlijker het is dat technische schuld ook zal toenemen.

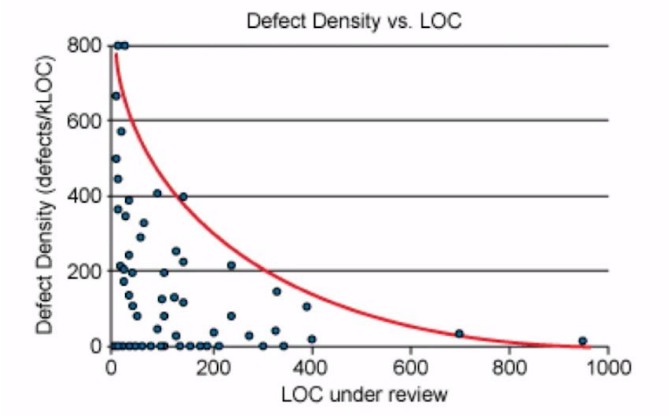

*figuur 23 - Voorstelling van technical debt*

Om de technische schuld te verkleinen, moet code wanneer mogelijk geoptimaliseerd worden. In een ideale wereld gebeurt dit proces van *refactoring* herhaaldelijk.

#### 3.3.1.4 Code reviews

Een ander middel om code zo kwaliteitsvol mogelijk te houden, is om code te onderwerpen aan *code reviews*. In de praktijk betekent dit dat wanneer een *pull request* wordt ingediend om een *merge* uit te voeren, dat bij voorkeur twee *code reviewers* aangeduid worden die de code zullen nalezen en vervolgens zullen goedkeuren of afkeuren. Dit gehele proces is ook gedetailleerd omschreven in het onderzoekstopic.

Cisco heeft een onderzoek gedaan naar het verband tussen de grootte van de *change* enerzijds, en het aantal gevonden defecten door *code reviewers* anderzijds.




*figuur 24 - resultaten onderzoek naar code reviews door Cisco*

Zoals bovenstaande grafiek aantoont, heeft de grootte van de *change* een drastische impact op het aantal gevonden defecten. Zo neemt het aantal gevonden defecten dramatisch af hoe meer code er te *reviewen* is. Het onderzoek van Cisco heeft dan ook aangetoond dat een *change* best beperkt wordt tot maximaal 400 lijnen code. Wanneer er toch meer code zou ingediend moeten worden, wordt deze beter opgedeeld in kleinere onderdelen. [9]

*Code reviews* kunnen op zich ook voor een verhoogde *cycle time* van *issues* zorgen. Het is dan ook goed om bijvoorbeeld te beperken hoeveel openstaande reviews een *developer* mag hebben vooraleer hij zich uitsluitend moet richten op het reviewen, of om *inbox zero days* in te voeren, waarop *developers* uitsluitend mogen reviewen.

### 3.3.2 Code Quality tools

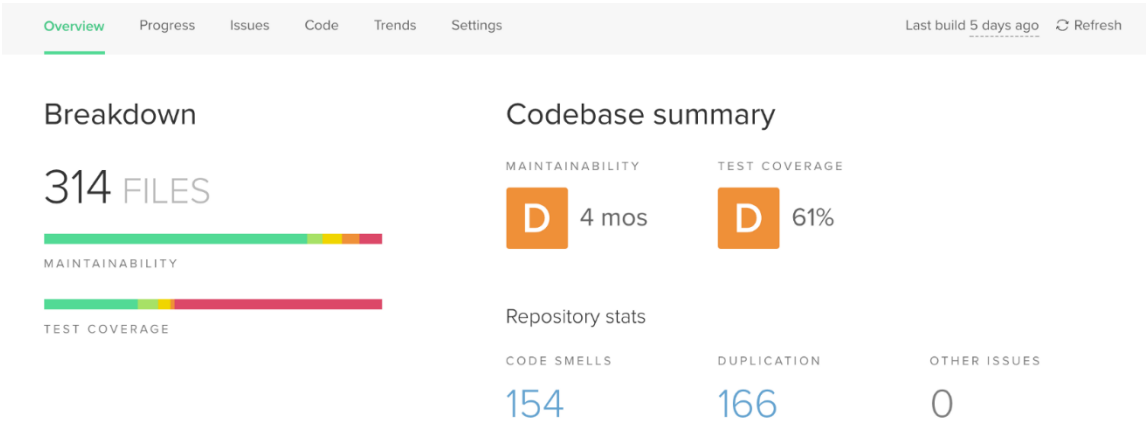
#### 3.3.2.1 Inleiding

*Code quality tools* analyseren, zoals eerder beschreven, de code en gaan na waar verbeteringen mogelijk zijn. Binnen Bewire werd er gewerkt met SonarQube, maar in het kader van de stageopdracht en de analyse van een geschikte *pipeline*, zijn mogelijke alternatieven ook onder de loep genomen.  Naast SonarQube zijn enkele waardige alternatieven geselecteerd en onderzocht, namelijk Code Climate, Codebeat en Codacy.

#### 3.3.2.2 Code Climate

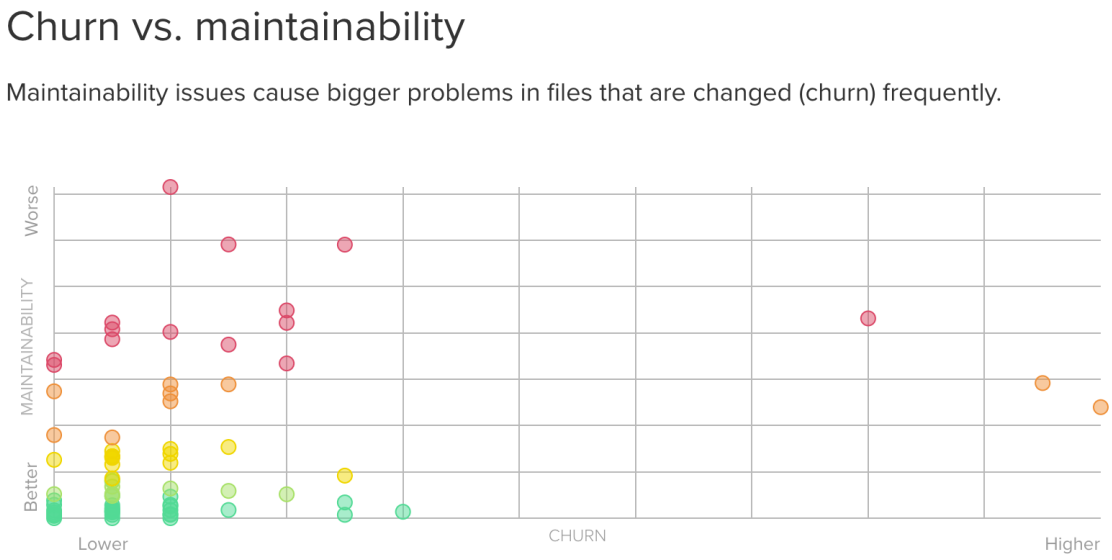
De eerste *tool* is veruit de meest gebruikte tool van allen. Code Climate scoort hoog op gebruiksvriendelijkheid en heeft bovendien *Integrated Development Environment (IDE)* integratie.

Het dashboard van Code Climate is erg vergelijkbaar met dat van SonarQube. Code krijgt een score van A tot F op *maintainability* en op *test coverage.*




*figuur 25 - Code Climate dashboard*

Een groot voordeel van Code Climate, is de mogelijkheid om testen te sorteren op bijvoorbeeld *test coverage.* Op deze manier kunnen de 'slechtste' stukken code eerst aangepakt worden. Daarnaast genereert Code Climate ook *trend charts.* In deze grafieken kan de gebruiker duidelijk zien of bijvoorbeeld de technische schuld toeneemt, of de test coverage al dan niet afneemt, enzovoort. Bovendien is er ook een rapport dat een afweging maakt tussen *maintainability en churn*. *Churn* geeft bestanden aan die frequent aangepast worden. Een dergelijk rapport kan de gebruiker dus aanzetten om heel gericht in te grijpen in de betrokken bestanden.




*figuur 26 - Churn vs. maintainability rapport van Code Climate*

Code Climate gebruikt enerzijds bekende *engines* om aan analyse te doen, zoals bijvoorbeeld ESLint. Anderzijds zet Code Climate zichzelf ook als platform voor *open-source engine development* in de verf. Code Climate heeft immers een grote *community* van *open-source developers* achter zich, waardoor voortdurend nieuwe *engines* ontwikkeld worden.

Vroeger was Jira-integratie mogelijk met Code Climate, maar deze functionaliteit bestaat helaas niet meer.

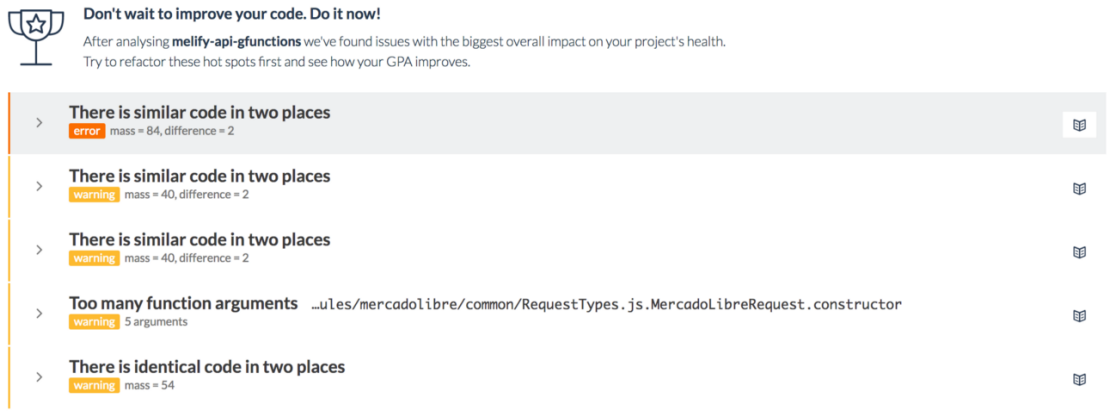
Een groot nadeel van Code Climate ten opzichte van SonarQube is echter dat Code Climate enkel een gratis versie heeft voor publieke *repositories*. Wanneer Code Climate wordt gebruikt voor *private repositories*, betaalt de organisatie $16.67 per maand, per gebruiker.

#### 3.3.2.3 Codebeat

Codebeat is de volgende tool die onderzocht is. Codebeat profileert zich op de markt als een *code quality tool* met een focus op *mobile development.*

Codebeat gebruikt een eigen algoritme om de codekwaliteit te bepalen. Bovendien is dit algoritme erg accuraat.

Een aantrekkelijke *feature* van Codebeat, is het *quick wins screen.* Op dit scherm worden de vijf resultaten van de analyse weergegeven die de grootste impact hebben om de algemene score van het project. Het aanpakken van deze vijf problemen zou het project dus drastisch kunnen verbeteren.

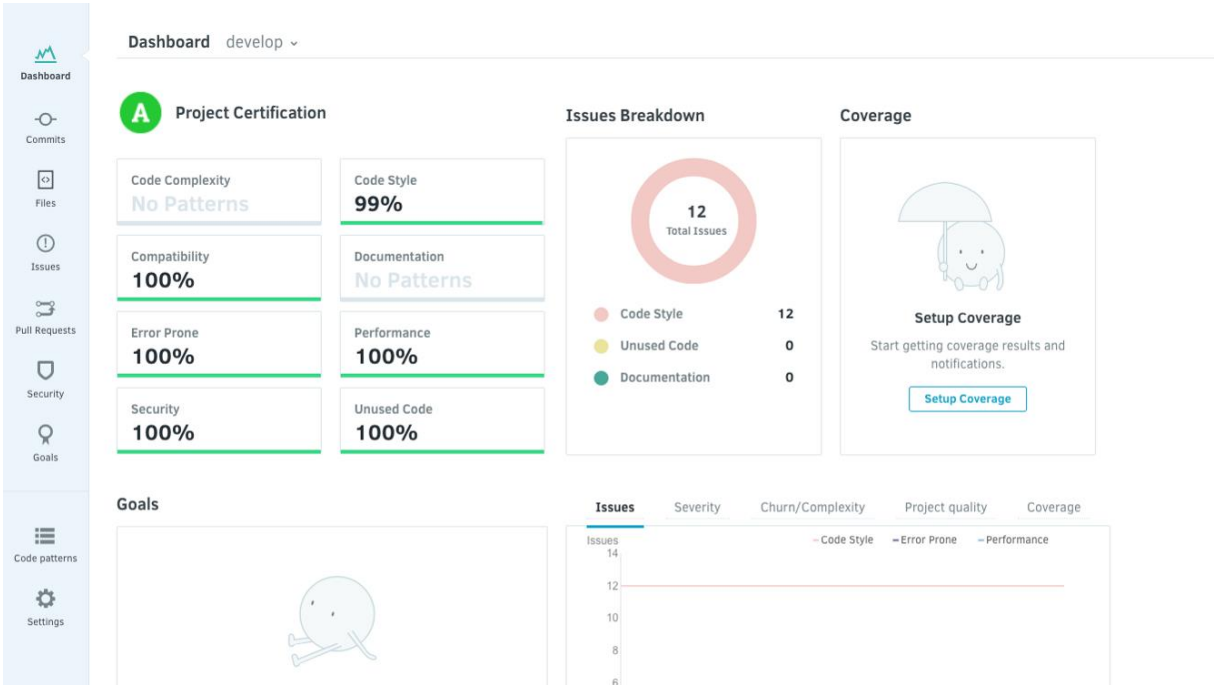



*figuur 27 - Quick wins van Codebeat*

Hoewel Codebeat dus erg aantrekkelijk lijkt, zijn er toch ernstige problemen, die een keuze voor Codebeat vrijwel onmogelijk maken. Zo is het bijvoorbeeld onmogelijk om een grenswaarde voor de minimale codekwaliteit of *test coverage* in te stellen vooraleer een *pull request* mag doorgaan. Daarnaast is het algoritme van Codebeat ook volledig gesloten. Concreet betekent dit dat er geen bijkomende plug-ins of *rules* kunnen toegevoegd worden. Daarnaast moet een organisatie die gebruik wil maken van Codebeat met *private repositories* $20 per maand, per gebruiker neerleggen.

#### 3.3.2.4 Codacy

Codacy is een *code quality tool* die de codekwaliteit beoordeelt aan de hand van acht categorieën. Bovendien kunnen ook projectdoelstellingen aan de hand van deze categorieën of op bestandsniveau opgesteld worden.



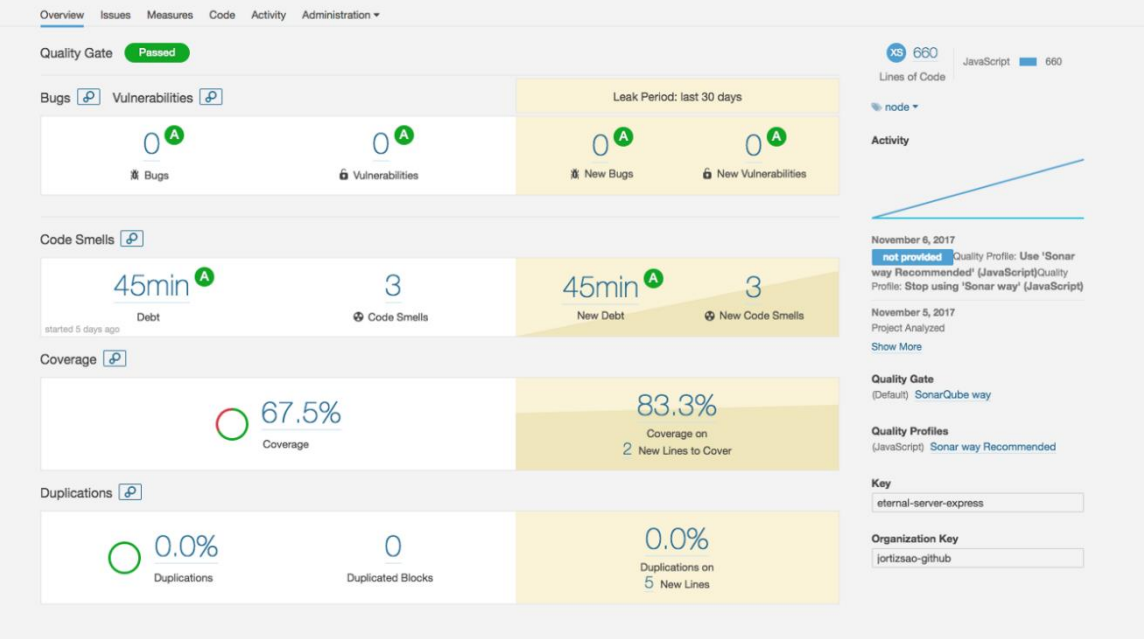

*figuur 28 - Dashboard van Codacy*

Codacy scoort hoog op veiligheidsanalyse en baseert zich daarvoor op de *Open Web Application Security Project* (OWASP) top tien van kwetsbaarheden. Daarnaast maakt ook gebruik van Hadolint voor analyse van Docker-bestanden. Hierdoor is Codacy een ideale toevoeging aan een *CI-pipeline.* Helaas is er ook aan Codacy een niet te verwaarlozen kostenplaatje verbonden. Een organisatie betaalt immers $18 per gebruiker per maand aan Codacy voor *private repositories*.

#### 3.3.2.5 SonarQube

SonarQube is momenteel al in gebruik bij Bewire. SonarQube is van alle *code quality tools* de meest krachtige. SonarQube heeft immers de grootste hoeveelheid *rules* en filters. SonarQube beschikt ook over *quality profiles* en *quality gates. Quality profiles* zijn een verzameling van regels waaraan de code moet voldoen om als kwaliteitsvol beschouwd te kunnen worden. Dit is enorm handig, aangezien er voor bepaalde projecten of bepaalde klanten verschillende profielen opgezet kunnen worden. *Quality gates* zijn grenswaarden waaraan code moet voldoen. Zo kan een gebruiker bijvoorbeeld instellen dat er minstens 85% *test coverage* mag zijn, alvorens een *build* mag slagen.

SonarQube is echter wel redelijk complex om op te zetten. Bovendien kan SonarQube ook niet zelfstandig de analyse uitvoeren, maar heeft hiervoor een CI-server nodig. Het grootste nadeel van SonarQube is echter dat de analyse pas na een *merge* uitgevoerd wordt, en niet daarvoor. Wanneer dit toch wenselijk zou zijn, moet er ook een licentie voor SonarQube aangekocht worden. De kostprijs van een SonarQube-licentie is echter moeilijk te bepalen. Een licentie is immers te verkrijgen vanaf €120, maar is volledig afhankelijk van de samenstelling van het team en het aantal lijnen code dat geanalyseerd moet worden. Eens dit aantal lijnen bereikt is, moet de keuze gemaakt worden tussen een meerprijs betalen voor verdere analyse, of het aantal lijnen code beperken.




*figuur 29 - SonarQube dashboard*

### 3.3.2.6 Besluit

Het is niet eenvoudig om een afweging te maken tussen bovengenoemde *code quality tools.* Wanneer de nadruk moet liggen op een zo laag mogelijke kostprijs en zoveel mogelijk controle op de code zelf, is SonarQube met gratis licentie de duidelijke winnaar. Het is bovendien ook een meerwaarde om voor bepaalde projecten een profiel te gebruiken dat op maat gemaakt is. Het is echter een enorm nadeel dat SonarQube de analyses pas uitvoert na een *merge.*

Op basis van klanttevredenheid en gebruiksvriendelijkheid scoort Code Climate dan weer het hoogst. Bovendien genereert Code Climate een aantal rapporten waardoor *developers* gerichter aan de slag kunnen om de kwaliteit te verhogen.

Wanneer de nadruk echter meer op security moet liggen en er ook controle nodig is voor de Docker-bestanden, is Codacy een aangeraden keuze.

### 3.3.3 Test management in Jira

#### 3.3.3.1 Test management tools

##### TM4J

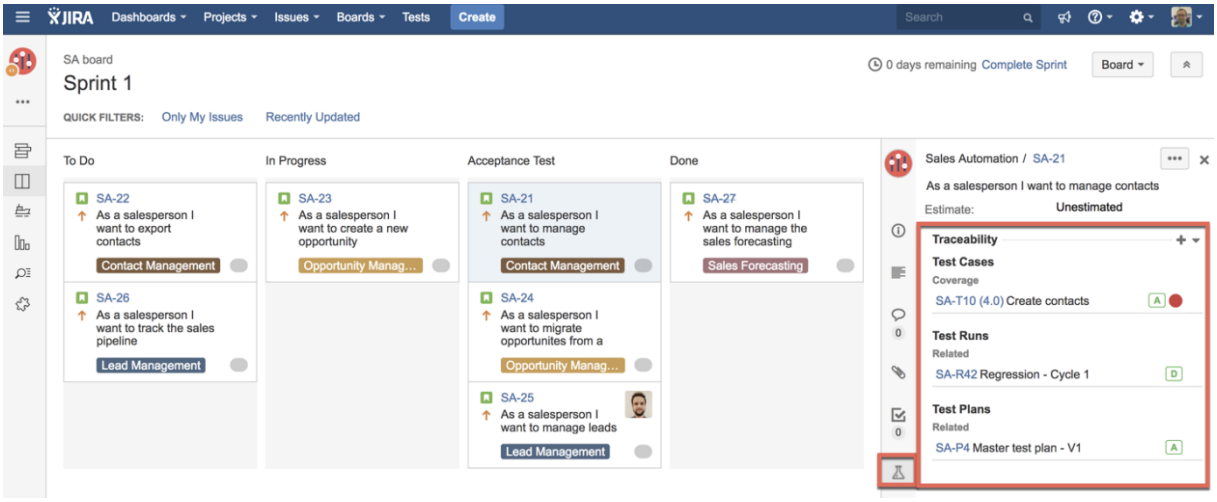
Test Management for Jira, oftewel TM4J, is de eerste *tool* die mogelijk deel kan uitmaken van de *software delivery pipeline* van Bewire.

Bewire heeft al enige ervaring met deze tool, maar toch waren zij op zoek naar een mogelijk beter alternatief. TM4J is immers niet zo eenvoudig in gebruik, en vraagt van de gebruiker toch enige kennis van QA.

TM4J creëert een bijkomend menu-item binnen een projectpagina op Jira, namelijk “tests”. Hier kan de gebruiker *test cases* aanmaken en uitschrijven, bijvoorbeeld aan de hand van *steps.* Vervolgens kunnen deze testen gekoppeld worden aan bestaande *issues* in Jira.

Het schrijven van testen in TM4J lijkt op het eerste zicht vrij omslachtig en tijdrovend. Bovendien lijkt de functionaliteit vanTM4J ook vrij beperkt. Aangezien Bewire geen intern QA-team heeft, is dit niet noodzakelijk een probleem, maar met het oog op mogelijke toekomstige ontwikkelingen, en het toenemende belang van QA, is dit een mogelijke beperking.

De kostprijs van TM4J, bedraagt $3.50 per gebruiker per maand.




*figuur 30 - Sprint board met TM4J*

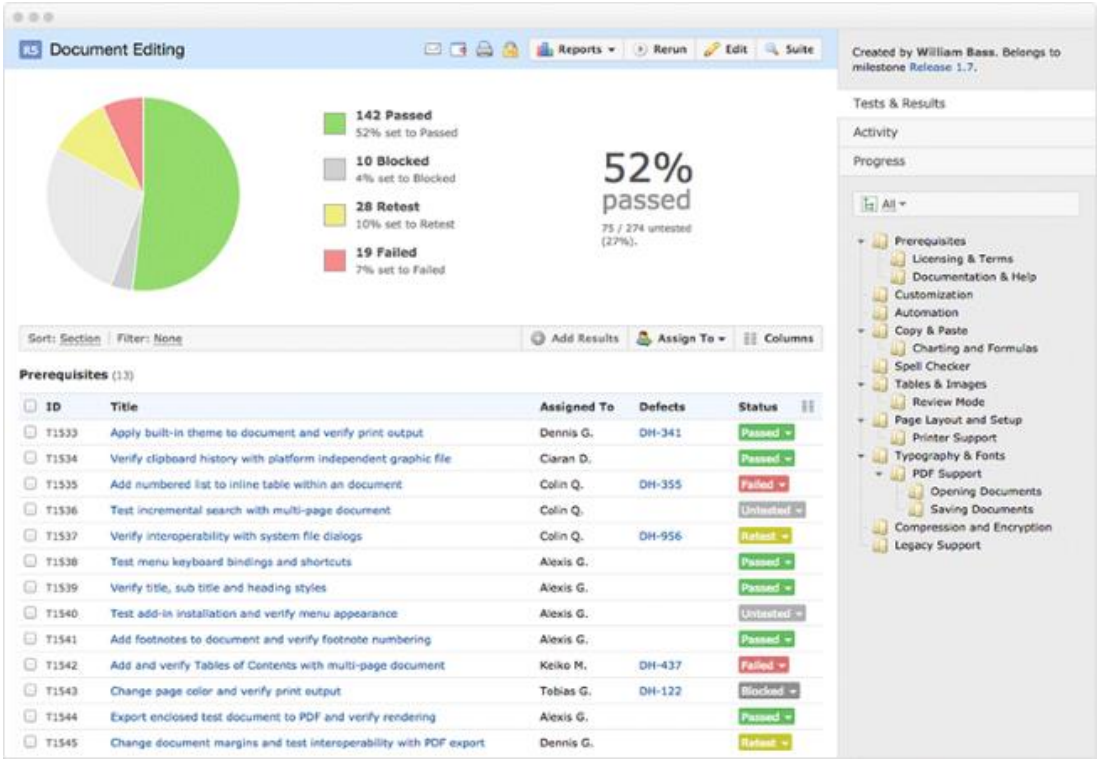
##### Testrail

Door persoonlijke ervaring, vond ik het ook de moeite om Testrail onder de loep te nemen, en na te gaan of Testrail een meerwaarde zou kunnen zijn voor Bewire.

Net zoals bij TM4J, is de functionaliteit van Testrail eerder beperkt, maar volstaat deze zeker om aan de vraag van Bewire te voldoen. Hier geldt echter dezelfde opmerking als voor TM4J. Aangezien de functionaliteit eerder beperkt is, is Testrail mogelijk niet toekomstbestendig.

Het grote nadeel van Testrail is echter dat, hoewel Testrail wel opgenomen in de Atlassian suite, het schrijven van de *test cases* niet rechtstreeks kan gebeuren in een Jira-project. Aan de andere kant is het wel zo dat Testrail volledig gratis is. Daarnaast kunnen er ook duidelijke rapporten gegenereerd

worden, wat zeker een meerwaarde is om projecten op te volgen en om te kunnen rapporteren naar klanten toe.



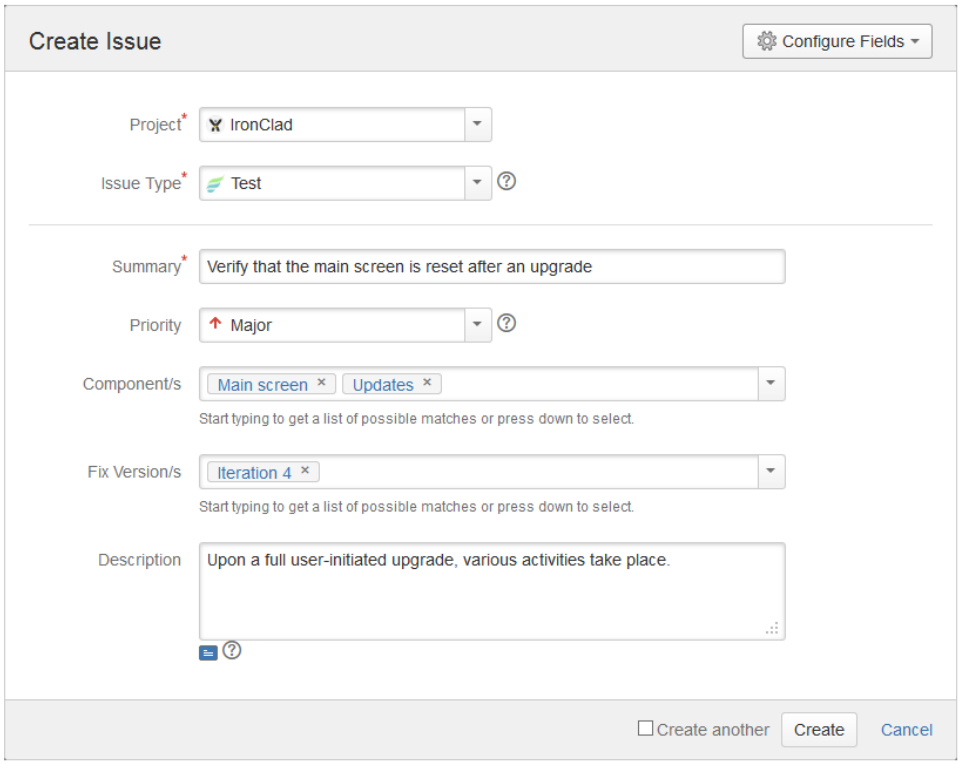

*figuur 31 - een dashboard in Testrail*

### Zephyr

Zephyr is een veelbelovende *tool,* en is bovendien ontwikkeld door SmartBear. SmartBear is een absolute marktleider op vlak van testing. Het bekendste product van SmartBear is, naast Zephyr, namelijk SoapUI (Pro).

In tegenstelling tot TM4J en Testrail, worden *test cases* in Zephyr rechtstreeks als issue types op het kanbanbord of sprintbord uitgeschreven. Dit verhoogt de zichtbaarheid van testresultaten voor het hele team, aangezien de rapportage eveneens rechtstreeks in Jira gebeurt. Het nadeel is echter dat dit snel onoverzichtelijk kan worden. Het is daarom aangeraden om met meerdere borden op een project te werken om de traditionele *issues* van de *test issues* te scheiden. Op deze manier worden tickets voor *development* niet bedolven onder een hele hoop *issues*, en blijft het overzicht bewaard.

Zephyr is enorm veelbelovend, maar is een rechtstreekse concurrent van Xray, de volgende *tool* die onderzocht is. Xray heeft meer functies, en is ook goedkoper. Hoewel beide tools momenteel dus zeker beantwoorden aan de noden van Bewire, lijkt Xray meer mogelijkheden te bieden voor een voordeligere prijs. Voor het gebruik van Zephyr zou Bewire bijvoorbeeld $4.25 per gebruiker per maand moeten neertellen. Het is wel belangrijk om er dan eveneens rekening mee te houden dat er nog bijkomende plug-ins moeten geïnstalleerd worden om bijvoorbeeld gerichte en duidelijk rapporten te kunnen exporteren. In werkelijkheid zal het kostenplaatje dus wat hoger liggen.

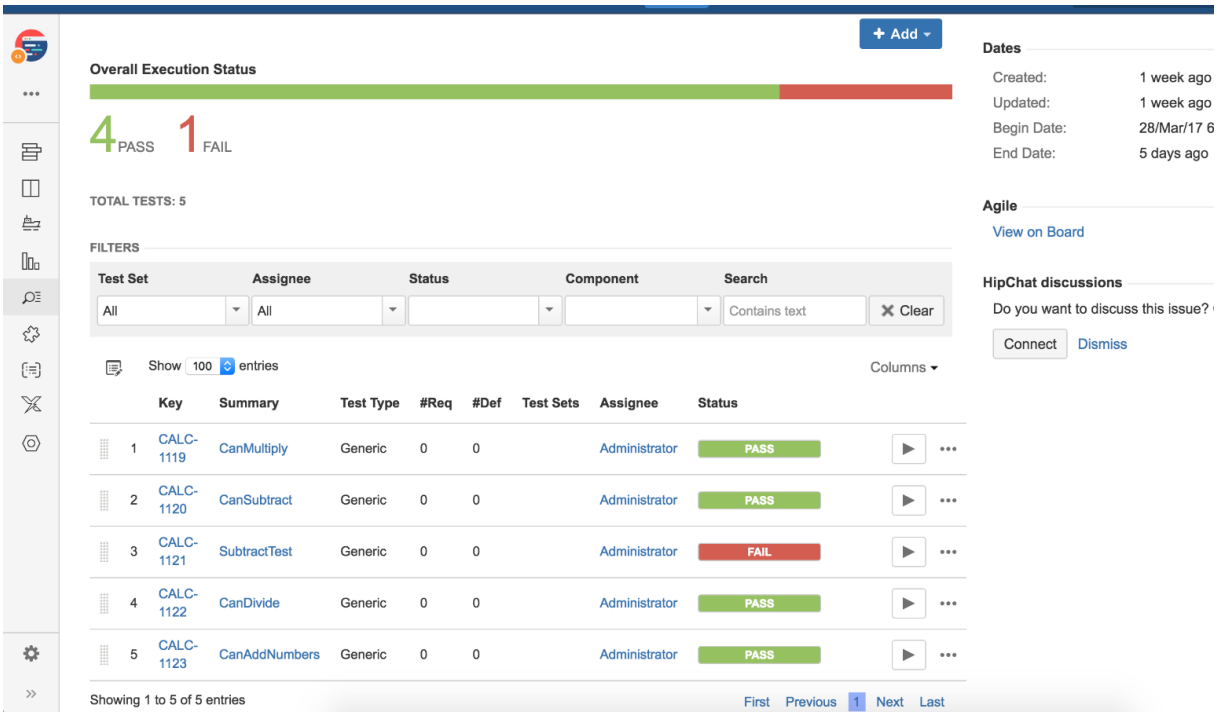



*figuur 32 - Een test issue in Jira met Zephyr*

### Xray

Net zoals Zephyr, werkt Xray met eigen *issue types* om *test cases* uit te schrijven. Deze *test issues* worden dus eveneens rechtstreeks op het bord zichtbaar, alsook de status van de verschillende testen.

Xray lijkt in werking erg op Zephyr, maar heeft toch nog enkele bijkomende krachtige functies. Zo is het eenvoudiger om testen te automatiseren met Xray. De krachtigste *feature* van Xray is ongetwijfeld de *native support* voor *Behavior-Driven Development* met Cucumber. Dit zou mogelijk in de toekomst of zelfs op korte termijn voor Bewire ook een meerwaarde kunnen betekenen. Zoals beschreven in het onderzoekstopic, kunnen Cucumber-testen immers geschreven worden door een business analist of een *product owner*, en kunnen de testen gewoon door de *developers* geschreven worden, zoals zij ook *unit tests* schrijven.




*figuur 33 - Dashboard binnen Jira met Xray*

### 3.3.3.2 Implementatie van Xray

De concrete implementatie en mogelijkheden van Xray worden omschreven in het onderzoekstopic. Hier wordt namelijk een usecase met Xray uitgewerkt in het kader van *Behaviour-Driven Development* en het schrijven van geautomatiseerde Cucumber-testen met integratie in de *continuous integration pipeline*.

## 3.4 Support & maintenance

### 3.4.1 Jira Service Desk

#### 3.4.1.1 Inleiding

In het kader van support en *maintenance,* had Bewire de vraag om een geschikt middel te onderzoeken om support aan klanten aan te bieden. Een helpdesk of servicedesk is hier ideaal voor.

#### 3.4.1.2 Probleemstelling

Klanten moeten *bugs* kunnen rapporteren, maar moeten ook voor andere vragen over de applicatie terecht kunnen bij Bewire, zeker eens een applicatie opgeleverd is en in gebruik is genomen. Hoewel er een aantal servicedesks op de markt zijn, is het ook belangrijk na te gaan hoe binnenkomende *tickets* opgenomen kunnen worden binnen de bestaande *workflows* van Jira. Als er bijvoorbeeld een *bug* gemeld wordt, en deze komt binnen bij support, dan moet dit ticket ook nog bij de juiste *developer* terechtkomen. Eens de *bug* opgelost is, moet dit ook correct doorgegeven worden aan support én aan de klant. Het is dus belangrijk om een servicedesk te selecteren die Jira-integratie heeft en liefst ook automatisatie kan realiseren om support *tickets* en jira-*tickets* te synchroniseren. In het kader van deze probleemstelling en andere implementaties van de Atlassian stack, wordt een implementatie van Jira Service Desk opgezet.

#### 3.4.1.3 Werkwijze en gerealiseerde implementaties

##### Soorten servicedesks met Jira Service Desk

Binnen Jira Service Desk kunnen verschillende types van servicedesks opgezet worden, afhankelijk van de noden van het bedrijf. Deze soorten zijn de volgende:

**a)** **External Service Desk**
Deze servicedesk voorziet support naar externe klanten toe. Hierbij wordt gebruik gemaakt van eenvoudige *workflows* gekoppeld aan een aantal *issue types.* Deze servicedesk sluit dus het meeste aan bij de noden van Bewire, waardoor er dus een concrete uitwerking is gebeurd.

**b)** Internal Service Desk

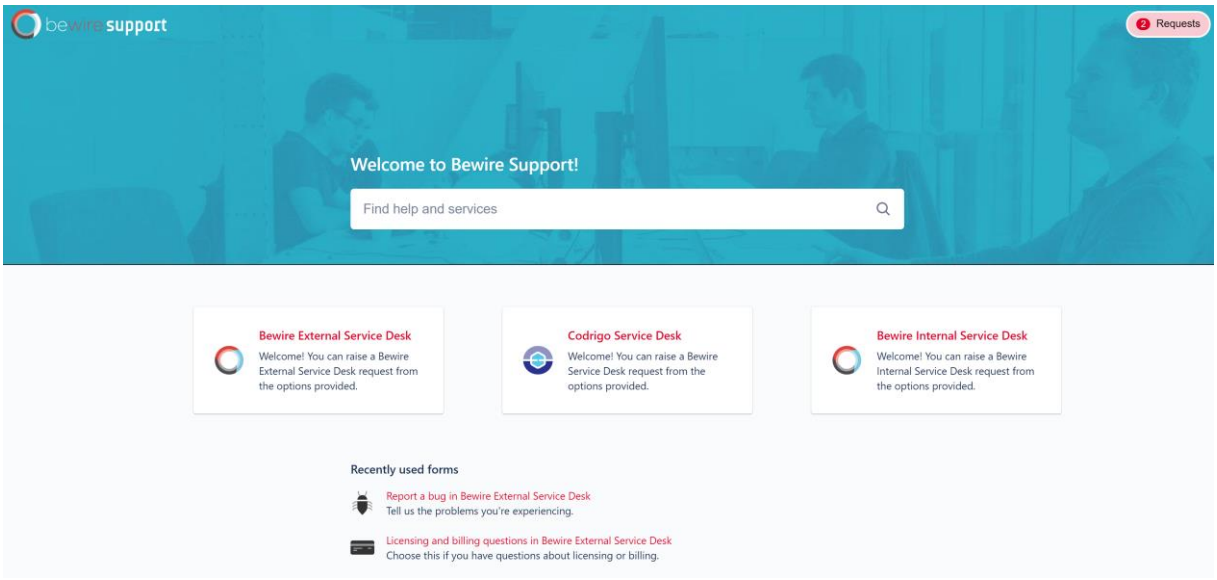
Zoals de naam aangeeft, is deze servicedesk geschikt om intern support te verlenen. Verder is de structuur gelijkaardig aan de *external service desk.*

**c)** **ITSM Service Desk**
Deze opgebouwd naar het referentiekader van de *Information Technology Infrastructure Library (*ITIL*)* en is tevens *ITIL-certified*. Deze servicedesk is in de praktijk dus iets uitgebreider en is meer complex. Het ITIL-certificaat is op dit moment geen meerwaarde voor Bewire, dus deze servicedesk valt buiten de *scope* van dit onderzoek.

##### Opzetten van supportkanalen

Klanten moeten via zo eenvoudig mogelijk een bug kunnen melden of een vraag voor support kunnen indienen. Jira Service Desk voorziet hier enerzijds support via e-mail, waarbij het e-mailadres en domein volledig te configureren valt. Zo kan er bijvoorbeeld gekozen worden voor *support@codrigo.be,* zonder bijkomende kost.

Verder kan er ook een klantenportaal opgezet worden waar klanten hun problemen kunnen melden aan de hand van een aantal categorieën. Door Confluence te linken aan de gekoppelde Jira-instantie, kan er ook een *knowledge base* opgebouwd worden, waardop bijvoorbeeld veelgestelde vragen beantwoord kunnen beantwoord worden en waar eveneens handleidingen voorzien kunnen worden.



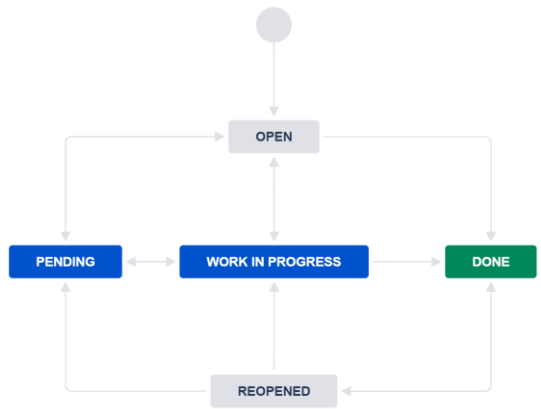

*figuur 34 - Het opgezette klantenportaal van Jira Service Desk*

Een belangrijke opmerking hier is dat er per Jira-instantie slechts één klantenportaal kan opgezet worden. Dit betekent dat de juiste permissies voor klanten moeten ingesteld worden, zodat zij enkel toegang hebben tot projecten waarin zij betrokken zijn.

### Analyse van issue types en workflows

Wanneer een klant een *ticket* opent voor een bepaalde vraag voor support, wordt er een bepaald *issue type* gekoppeld aan deze vraag. Dit kan een *bug* zijn, indien de klant hier een melding van maakt, een *support issue type*, voor een algemene vraag, en tot slot een *feature*, als een klant een nieuwe functionaliteit wil voorstellen. Uiteraard kunnen deze *issue types* beperkt worden, of uitgebreid worden met bestaande of eigen *issue types.* Uiteraard moeten die types dan ook nog gekoppeld worden aan de juiste *request types.*

In deze paper worden enkel bovengenoemde *issue types* behandeld. Onderstaande workflows zijn gekoppeld aan deze types.

| | |
|---|---|
| Bug, Feature |  |



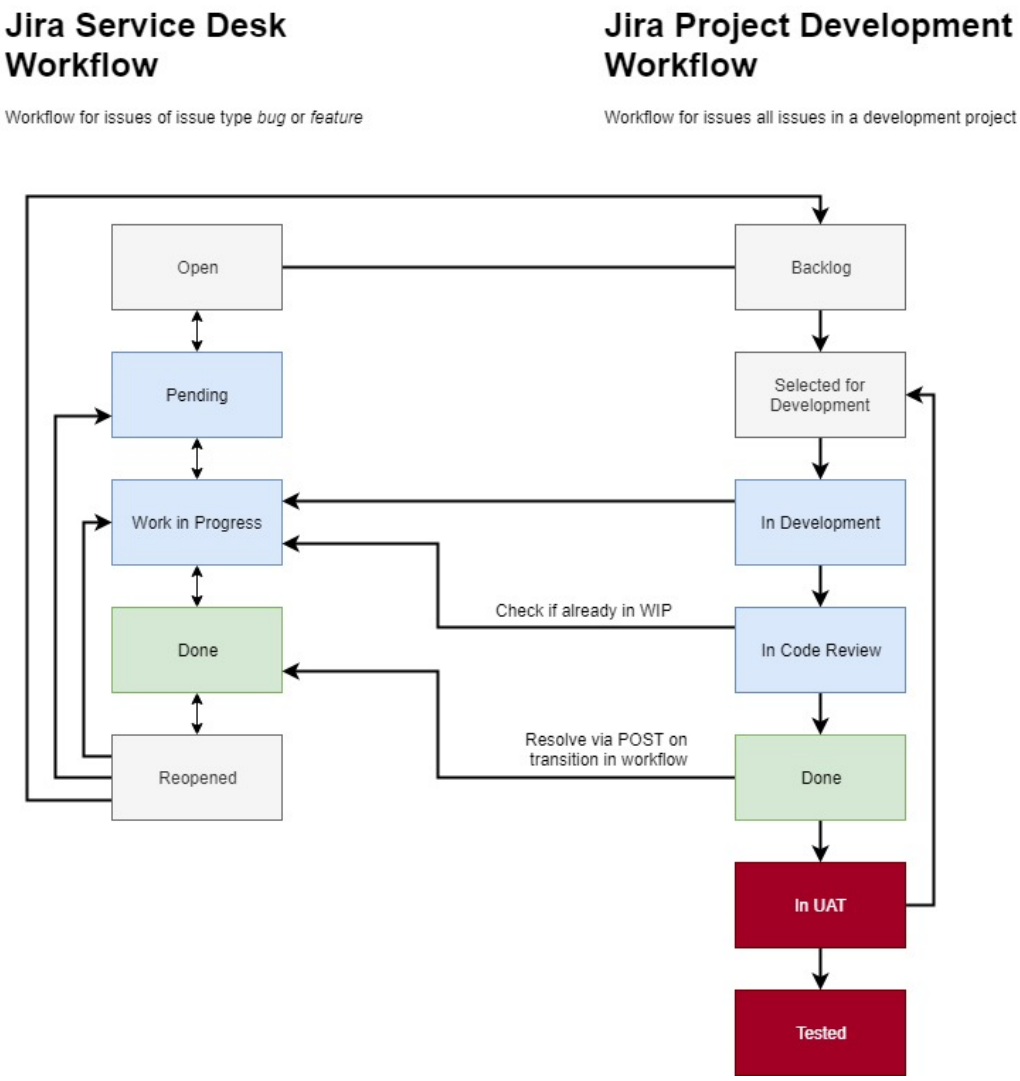
*figuur 35 - Jira Service Desk workflow voor bugs en features*

| | |
|---|---|
| Support |  <br>*figuur 36 - Jira Service Desk workflow voor support issue types* |

### Automatisatie en synchronisatie met een Jira-project

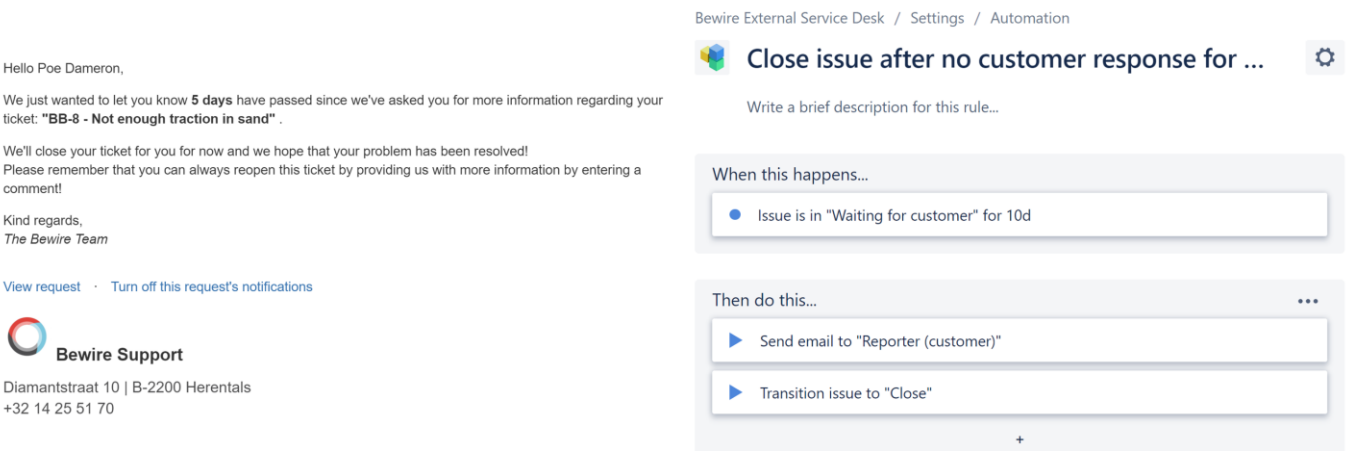
Jira Service Desk voorziet standaard de functionaliteit om diverse automatisatieregels te schrijven. Dit kan gaan van het automatisch verplaatsen van Jira Service Desk *issues* op basis van de transities van gelinkte issues in een Jira-project, het automatisch afsluiten van onbeantwoorde *tickets* of het automatisch versturen van e-mails naar klanten die een ticket hebben aangemaakt of naar de *assignee* van een ticket waarbij een *SLA-breach* dreigt te gebeuren.

Op basis van de omschreven *workflows* kan er dus automatisatie en synchronisatie gerealiseerd worden tussen een servicedeskproject en een Jira-project waarin de ontwikkelaars werken, zoals gemodelleerd in het schema op de volgende pagina.




*figuur 37 - Synchronisatie door automatisatieregels tussen een servicedesk en Jira-project*

Een ander voorbeeld van mogelijke automatisatie is het versturen van een e-mail naar een klant die enige tijd niet geantwoord heeft op een openstaand ticket waarbij bijkomende informatie noodzakelijk is. Merk op dat de mails gepersonaliseerd kunnen worden. Het is immers mogelijk om enerzijds Cascading Style Sheets (CSS) te gebruiken om de e-mail visueel aan te passen aan de bedrijfsstijl, en anderzijds kan er gebruik gemaakt worden van JQL om de klant bij naam aan te spreken en om naar bepaalde velden van het ticket te verwijzen. In de afbeelding op de volgende pagina is een dergelijk voorbeeld uitgewerkt.




*figuur 38 - Opbouw automatisatieregel voor e-mail met bijhorend resultaat*

### Jira Service Desk agents & collaborators

Om Jira Service Desk aan te kopen, betaalt de organisatie enkel de kostprijs van het aantal actieve *agents.* Een agent is een Jira-gebruiker die support *tickets* moet kunnen bewerken en afhandelen. Elke Jira-gebruiker die geen *agent* is, kan echter wel nog de support tickets lezen. *Developers* kunnen bijvoorbeeld wel meewerken aan support *tickets* en kunnen ook alle informatie van een ticket lezen, maar zelf kunnen zij dus geen aanpassingen doen aan het *ticket*, het *ticket* van status veranderen of afsluiten. Daarnaast kunnen zij ook geen aanpassingen doen aan bijvoorbeeld de *request types,* het klantenportaal, enzovoort. Er moet dus goed nagedacht worden over wie een *agent* hoort te zijn en wie enkel tickets moet kunnen lezen. Dat kan immers de kostprijs dramatisch beïnvloeden.

# II. Onderzoekstopic

# 1 Inleiding

## 1.1 Aanleiding

In het kader van de nieuwe start-up binnen Bewire, moet een gepaste *software delivery pipeline* omschreven worden. De focus ligt op kleinere projecten binnen mobile development. Bewire heeft echter al een *workflow* die op het eerste zicht als ideaal omschreven zou kunnen worden. Toch ontbreekt de gewenste snelheid in het ontwikkelingsproces, bijvoorbeeld omdat de ontwikkelaars nog veel aandacht en tijd moesten besteden aan het updaten van Jira-tickets, het voorbereiden van bepaalde *workflows* in Jira, het manueel *releasen* en *deployen* in de stores. Een *DevOps-pipeline* kan een mogelijke oplossing zijn voor de voorgelegde problemen. DevOps wordt echter al te vaak uitsluitend technisch geïnterpreteerd. Bovendien ligt de focus veelal uitsluitend op development- of operations-processen, en valt projectmanagement vaak uit de boot. Projectmanagers plukken immers vaak passief de vruchten van de enorme verbeteringen in *development* en *operations*. Vandaar dat dit onderzoek dieper ingaat op aspecten van DevOps die niet exclusief verbonden zijn aan technische specificaties, of toepassingen die uitsluitend gericht zijn op *development* en *operations*.

## 1.2 Stand van zaken

DevOps is een *hot topic.* Er zijn al tal van publicaties geschreven en onderzoeken uitgevoerd naar mogelijke DevOps-toepassingen. Een aantal relevante bronnen die gebruikt worden als basis van dit onderzoek, zijn terug te vinden in de bibliografie.

Het is echter vaak zo dat DevOps wordt geprezen door de concrete verbeteringen binnen de *development- en operationsprocessen.* Uiteraard vergroot ook de productkwaliteit hier zienderogen. De nadruk van DevOps ligt in veel wetenschappelijke artikelen echter vaak op technische specificaties en de fysieke opbouw van *pipelines.* Projectmanagers plukken hier uiteraard ook de vruchten van, maar toch zijn er veel misvattingen over wat DevOps is, en hoe er een concrete invulling kan zijn voor projectmanagers. Bovendien is het doel hier ook om de waarde van DevOps voor kleine teams en kortlopende projecten te onderzoeken.

## 1.3 Scope

De resultaten van dit onderzoek moeten uitsluitend toepasbaar zijn op projectmanagement of op de algehele projectkwaliteit, zonder daarvoor een uiterst technische implementatie te zijn. Bovendien moeten de resultaten ook gericht zijn op kortlopende projecten met een focus op mobile *development*. Dat betekent dat er bijvoorbeeld geen toepassingen mogen onderzocht worden waar een complexe infrastructuur vereist is.

Naast de conclusies die uit het literatuuronderzoek, worden er ook praktische implementaties omschreven, zoals geïmplementeerd tijdens de experimentele onderzoeken, en uitsluitend als die gunstige resultaten hebben voortgebracht.

## 1.4 Probleemstellingen

We kunnen volgende probleemstellingen kaderen:

a) Er zijn veel misvattingen over DevOps; DevOps wordt vaak als een technisch iets beschouwd. De focus ligt veelal op fysieke *pipelines*.
b) De focus bij DevOps ligt vaak uitsluitend op *development* en *operations.*
c) De focus ligt vooral op productkwaliteit en niet op projectkwaliteit.
d) De baten voor projectmanagement worden vaak uitsluitend beschouwd als een gevolg van een goede fysieke *pipeline* en DevOps-toepassingen binnen *development* en *operations.*
e) Het implementeren van DevOps schrikken kleinere teams met kortlopende projecten af, omdat er vaak een verkeerde interpretatie van DevOps is, zoals:
    (i) *DevOps pipelines* berusten altijd op een complexe infrastructuur, waarvoor een *operations team* nodig is.
    (ii) De wens om *alle* toepassingen van DevOps gelijktijdig te implementeren.
    (iii) DevOps is enkel zinvol voor grote organisaties of grote projecten.
    (iv) DevOps is uitsluitend een verzameling van tools

## 1.5 Onderzoeksvraag

Naar aanleiding van bovengenoemde probleemstellingen, wordt onderzoek verricht naar het verbeteren van projectmanagement in organisaties voor *mobile development* aan de hand van verschillende principes van DevOps. De onderzoeksvraag luidt dus als volgt:

**“Hoe kunnen DevOps-principes ingezet worden om de kwaliteit van projectmanagement en de projectkwaliteit te verbeteren in kortlopende projecten voor mobile development?”**

# 2 Methodologie

## 2.1 Soort onderzoek

Om een antwoord te krijgen op de onderzoeksvraag en bijkomende beschrijvende vragen, zal een kwalitatief onderzoek uitgevoerd worden. Een deel van dit onderzoek zal immers gebaseerd zijn op een interpretatie van de DevOps-ideologie, en ervaringen van mogelijke implementaties in de *software delivery pipeline*, en niet uitsluitend op feiten en cijfers. Daarnaast zal ook een kwantitatief onderzoek uitgevoerd worden, om de resultaten uit het kwalitatief onderzoek te ondersteunen met cijfers en feiten.

## 2.2 Onderzoeksmethoden

Om tot de gewenste resultaten te bekomen zal enerzijds een literatuuronderzoek gedaan worden, eveneens aan de hand van relevante cases. Daarnaast zullen ook bepaalde experimentele onderzoeksfases plaatsvinden. Zo zullen bepaalde delen van een *software delivery pipeline* opgezet worden en

# 3 DevOps: Een theoretisch kader

## 3.1 Wat is DevOps?

Vooraleer praktische implementaties van DevOps in het kader van projectmanagement en projectkwaliteit omschreven worden, is het noodzakelijk te begrijpen wat DevOps eigenlijk is.

DevOps is een samenvoeging van het woord '*development*' en het woord '*operations.*' DevOps is een methode om op een *agile* manier software te ontwikkelen, maar gaat eigenlijk nog een stap verder dan "*agile."* Het gaat bij DevOps immers om de samenwerking tussen de softwareontwikkelaars en het *operations*-team. Beiden zijn immers verantwoordelijk dat het gewenste eindresultaat opgeleverd kan worden. In feite integreert DevOps dus beide teams om samenwerking en productiviteit te verbeteren. Naast deze samenwerking, zijn er nog een aantal principes die gebonden zijn aan DevOps en die in het volgende onderdeel toegelicht worden. [11]

## 3.2 Welke DevOps-principes zijn er?

Er bestaan heel wat definities van DevOps, maar toch blijft de essentie van DevOps vaak open voor interpretatie. Vandaar dat er vaak gewerkt wordt met een aantal principes die eigen zijn aan DevOps, om het begrip te kunnen kaderen. Hoewel er ook geen consensus is over welke principes DevOps definiëren en veel bedrijven dus een eigen invulling geven aan DevOps, zijn er een aantal principes die in deze paper weerhouden worden, namelijk:

f) Accountability, ownership & end-to-end responsibility
g) Collaboration
h) Automation
i) Continuous integration
j) Fast & reliable deploys with continuous delivery
k) Continuous Improvement

### Accountabililty, ownership & end-to-end responsibility

Met *accountability* en *end-to-end responsibility* wordt doorgaans bedoelt dat zowel *development* als *operations* een gedeelde verantwoordelijkheid hebben tijdens het gehele *software delivery proces.* Dit betekent dat beide teams een idee hebben van wat er allemaal speelt tijdens elke fase van de *software development life cycle.* In het kader van dit onderzoek wordt dit kader zoveel mogelijk verbreedt, namelijk om niet enkel de ontwikkelaars of *operation engineers* in deze verantwoordelijkheid te betrekken, maar ook andere profielen, zoals projectmanagers en testers. Uiteraard is het niet de bedoeling om de taaklast van elk teamlid uit te breiden, of om bijkomende verantwoordelijkheden op elk individu te storten, maar wel om

### Collaboration

Binnen DevOps is het concept van *collaboration* of samenwerking cruciaal. Het team komt samen om problemen van eender welke aard aan te pakken. Het is dus belangrijk dat er gepraat wordt en dat er begrip wordt opgebracht voor ieders verantwoordelijkheden, taken, enzovoort. Er wordt zo transparant mogelijk gewerkt, en er wordt actief ingezet om de betrokkenheid binnen het team te vergroten.

### Automation

DevOps is gericht op snelle opleveringen. Het is dus logisch om zoveel mogelijk te gaan automatiseren. Hoe meer handelingen automatisch gebeuren, hoe minder menselijke input nodig is. Dat betekent dat er enerzijds minder kans is dat menselijke fouten in het proces geïntroduceerd worden en dat anderzijds de werklast van de teamleden verminderd. Zij kunnen hun tijd besteden aan andere aspecten van hun job, waardoor er meer werk gedaan kan worden, en waardoor hopelijk de tevredenheid van de werknemers verhoogt.

### Continuous Integration (CI)

Met CI wordt de gebouwde code getest. Zie het hoofdstuk over CI met Bitbucket Pipelines voor een meer volledige omschrijving van CI en een praktische uitwerking met binnen de Atlassian stack.

### Fast & reliable deploys with Continuous Delivery (CD)

Na CI is CD de volgende stap. CD is de mogelijkheid om *changes*, ongeacht hun aard, snel in productie of rechtstreeks in de handen van de gebruikers te krijgen. Bovendien moet dit op een snelle, veilige en duurzame manier. Het doel is dus het optimaliseren van *deployments.* Het is dus een vereiste dat de code steeds in een *deployable* state is, zelfs al worden dagelijks talloze veranderingen aan de code toegevoegd. [12]
Hoewel projectmanagers onrechtstreeks baat kunnen hebben bij CD, ligt de nadruk van CD voornamelijk op *deployment.* Vandaar dat een toepassing van CD voor het bevorderen van projectmanagement als *out of scope* beschouwd worden binnen deze paper.

### Continuous Improvement

Veel organisaties die met DevOps willen beginnen, trappen in de valkuil dat bovenstaande principes allen onmiddellijk en volledig geïmplementeerd moeten worden. Dit is echter niet het geval. Een DevOps-cultuur moet steeds pragmatisch bekeken worden binnen een bedrijf. Welke gedachtegangen, middelen of methodieken zijn zinvol voor de organisatie in kwestie? Bovendien worden organisaties ook voortdurend geconfronteerd met veranderingen. De noden van klanten veranderen, de bestaande technologieën vernieuwen, de maatschappij evolueert, enzovoort. DevOps is een praktijk die zich steeds moet blijven aanpassen en inspelen op de noden van de organisatie en de maatschappij daarrond. Daarom is het belangrijk dat er voortdurend gezocht wordt naar mogelijke verbeteringen en manieren om te innoveren.

# 4 DevOps toegepast binnen de Atlassian suite

## 4.1 Inleiding

In dit hoofdstuk worden alle resultaten, die betrekking hebben tot de Atlassian *tools*, gerapporteerd. Als er wenselijke, concrete implementaties zijn voortgevloeid uit de experimentele onderzoeken, worden deze hier eveneens uitgeschreven, zodat deze kunnen geproduceerd worden in andere omgevingen. Waar mogelijk wordt dit in de vorm van een handleiding gedaan.

De Atlassian suite, en vooral Jira, is hét projectmanagementmiddel bij uitstek. Zelfs zonder enige bijzondere configuraties, worden, bij correct gebruik, veel DevOps-principes toegepast. Het is vrijwel ondenkbaar om tegenwoordig zonder Jira of een gelijkaardige *tool* te werken binnen projecten.

Het is echter niet de bedoeling om met dit onderzoek een handleiding voor Jira te produceren. Enkel configuraties en methodieken die een meerwaarde kunnen zijn voor projectmanagement vanuit een DevOps-optiek komen aan bod. Vandaar dat een algemene kennis van Jira wel verwacht wordt.

## 4.2 Doelstellingen

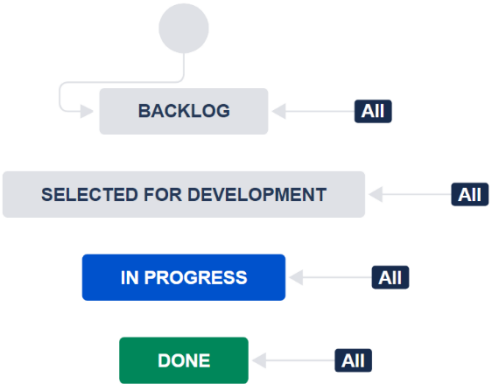
Door DevOps-principes toe te passen binnen de Atlassian *tools*, kunnen we bepaalde taken binnen projectmanagement vergemakkelijken. Bovendien kunnen we er op deze manier ook voor zorgen dat bijgevolg de project- en productkwaliteit verhoogt. Sterker nog, we kunnen tal van doelstellingen bereiken. Hier volgen alle doelstellingen op een rijtje:

a) Correctheid, volledigheid en traceerbaarheid van issues vergroten
b) Ontwikkelaars ontlasten door:
c) De tijd die ze spenderen aan tickets verkleinen
d) De tijd om te programmeren vergroten
e) De traceerbaarheid van hun werk vergroten
f) Feedback sneller, gerichter en correcter te geven
g) Betere rapportages genereren
h) De mogelijkheid tot samenwerking vergroten
i) Correctere code produceren
j) Code altijd in een *shippable state* houden

Bovenstaande doelstellingen zullen gerealiseerd worden door enerzijds automatisatie door te drijven binnen de Atlassian *tools*, en anderzijds door *Continous Integration (CI)* te realiseren. In volgende onderdelen van dit hoofdstuk zal dus gedetailleerd beschreven worden hoe hiermee is omgesprongen.

## 4.3 Jira Workflows

Een *workflow* binnen Jira is een verzameling van statussen en gekoppelde transities waardoor een *issue* zich kan verplaatsen. Concreet zal dit zich vooral uiten in de voorstelling van het kanbanbord waarop tickets zichtbaar zullen zijn, en waarop het team zal werken.

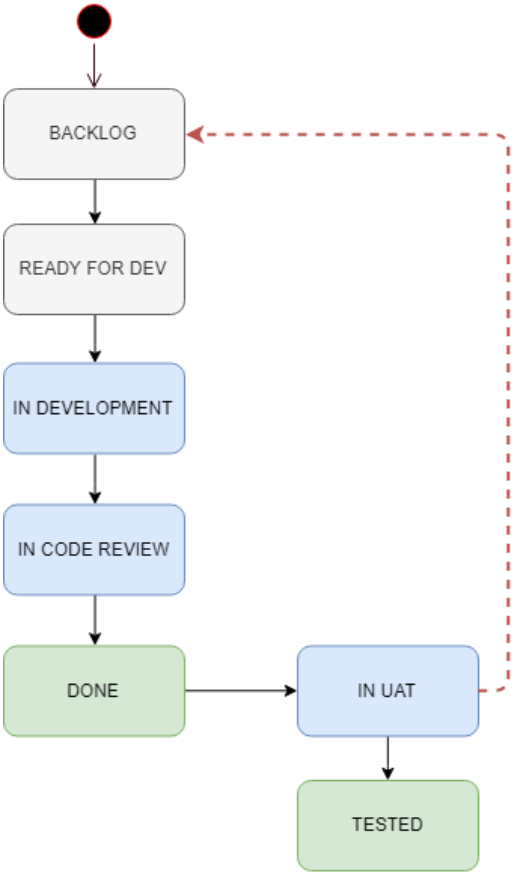



Op de figuur links is de standaard *Jira workflow* te zien. Wanneer een nieuw project gestart wordt, zal Jira automatisch deze *workflow* koppelen aan het project.

Voor eenvoudige projecten, kan deze workflow volstaan, maar meestal is een uitbreiding wenselijk, zeker met het oog op DevOps. Zo willen we bijvoorbeeld ook gaan werken met statussen waarin we kunnen aangeven dat een bepaalde opgeleverde *feature* momenteel onderworpen is aan *User Acceptance Tests (UAT)* en wanneer een bepaalde feature effectief goedgekeurd is. Daarnaast moet er ook gewerkt worden met *code reviews*, om ervoor te zorgen dat code zo goed mogelijk geschreven is.

*Figuur 39 - Standaard Jira workflow*

Voor kortlopende projecten kunnen we dan een *workflow* gebruiken zoals getoond in de volgende figuur.




We onderscheiden volgende statussen in deze workflow:

a) Backlog
b) Ready for development
c) In development
d) In code review
e) Done
f) In UAT
g) Tested

Merk op dat *backlog* en *selected for development* statussen zijn die worden beschouwd als *to do,* afgebeeld met de grijze kleur. Dit betekent dat er nog niet actief gewerkt wordt aan deze issues op het moment dat zij zich in een van deze statussen bevinden.

*In development, in code review* en *in UAT* zijn actieve statussen, die beschouwd worden als *in progress.* De groene statussen, *done* en *tested* zijn statussen die als afgerond beschouwd worden in respectievelijke stadia van het ontwikkelingsproces.

*Figuur 40 - Geselecteerde workflow voor kortlopende projecten*

## 4.4 Bitbucket

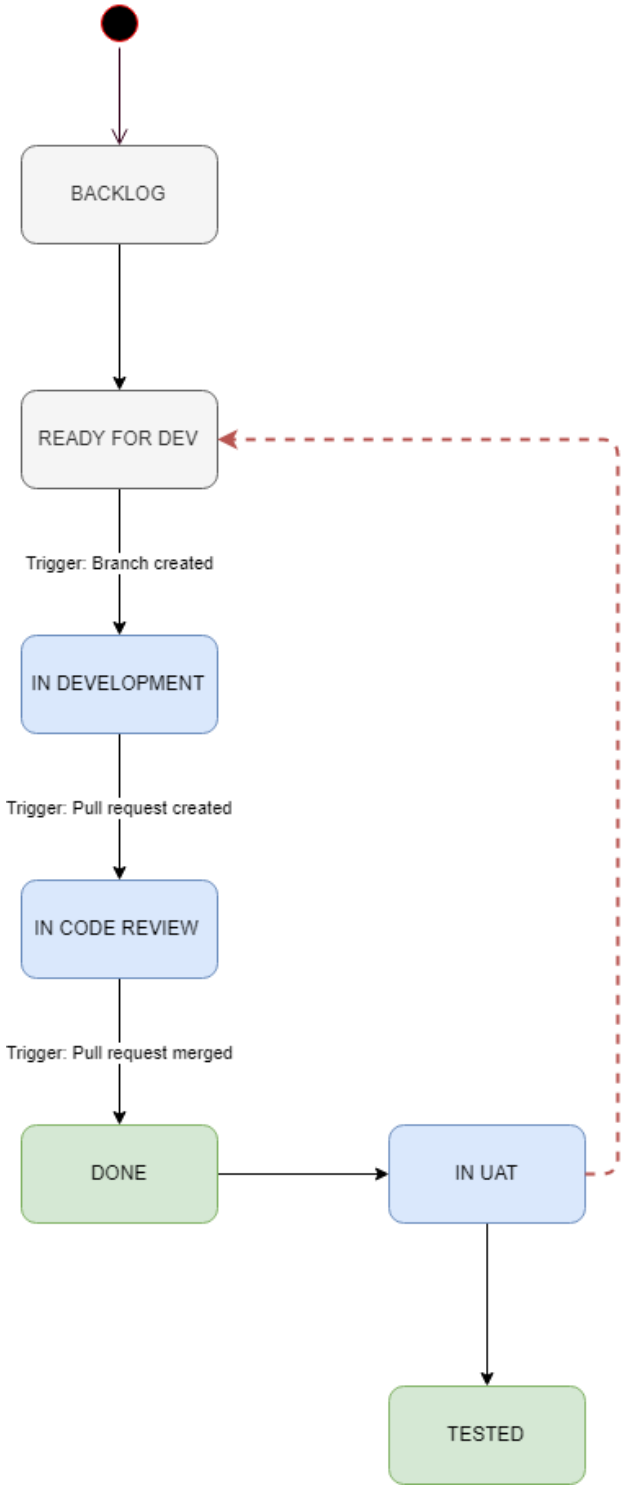
Bitbucket is eveneens een tool van Atlassian, en wordt gebruikt om aan *repository management* te doen. Aangezien Bitbucket uitgebracht is door Atlassian, is een diepgaande integratie mogelijk met Jira en andere Atlassian *tools*.

Om de doelstellingen te bereiken, is het dus noodzakelijk om de code te koppelen aan Bitbucket. Bitbucket moet vervolgens gekoppeld worden aan het gewenste Jira-project. Laat ons voor de verdere stappen in dit onderzoek, veronderstellen dat deze configuratie gebeurd is.

## 4.5 Workflowautomatisatie met Bitbucket-integratie

Om automatisatie binnen de workflow van Jira te realiseren, werken we met *triggers***.**

Op volgende figuur zijn de *transition branches* aangegeven met de te implementeren *triggers*.




*Figuur 41 - Jira workflow met automatisatietriggers*

Laat ons stellen dat de *requirements* en *scope* van het project al duidelijk vastgelegd zijn. De *backlog* is opgesteld door de projectmanager en zijn team. De *workflow*, zoals voorgesteld in Figuur 41 zit dan als volgt in elkaar:

### Van backlog naar ready for development

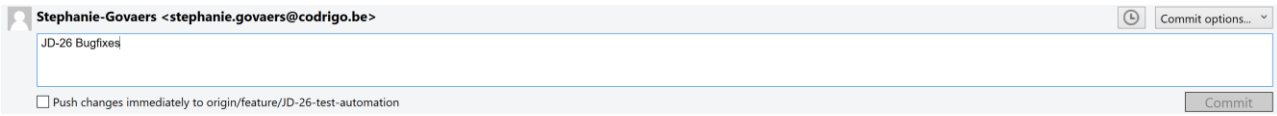
De *issues* die relevant zijn voor de sprint worden, na overleg en goedkeuring van de projectmanager geselecteerd en manueel naar *ready for dev* verplaatst. Dit is een stap die in principe eenmalig moet gebeuren bij de aanvang van een nieuwe sprint. In uitzonderlijke gevallen, wanneer *development* bijvoorbeeld voor op schema is, kunnen bijkomende *issues* eveneens manueel van de *backlog* naar *ready for dev* verplaatst worden. Het verplaatsen van issues van de *backlog* naar *ready for dev* tijdens een sprint moet echter beperkt worden, want dit kan problemen met zich meebrengen in de rapportages van de *sprint progress* of *resource planning*. Een projectmanager moet zijn team dus goed kennen en de *workload* die tijdens een sprint kan opgenomen worden moet zo nauwkeurig mogelijk vooraf ingeschat worden.

### Van ready for development naar in development

Eens een *issue* geselecteerd is om ontwikkeld te worden, kunnen ontwikkelaars een issue toe-eigenen om aan de slag te gaan. De issue wordt binnen Jira toegekend aan de gepaste *developer*, die op zijn beurt vanuit het gerelateerde ticket binnen Jira een *feature-branch* aanmaakt in de *repository* met behulp van Bitbucket. Door de *branch created-trigger* binnen de *workflows* van Jira worden *issues* automatisch verplaatst naar *in development*. Bij de creatie van een *branch* kan de ontwikkelaar meteen naar *code check-out* gaan via command line of via de *User interface (*UI) van Sourcetree.

### Van in development naar in code review

Het ontwikkelproces gaat nu van start voor de geselecteerde *issue*. Eens de code is geschreven, kan de ontwikkelaar de code ofwel via *commit* op de huidige *branch* opslaan, of, als de code werkend en volledig is, een *pull request* indienen, om de code terug te *mergen* met de *develop branch*. Dit proces kan via *command line* of via Sourcetree. Sourcetree is een Git Graphical User Interface (GUI), en geeft dus een visuele voorstelling van de repository weer. Hoe dan ook, het is cruciaal dat de ontwikkelaar de *Jira-ID-code* van het gerelateerde ticket meegeeft in zijn *commit message*. Op basis van deze ID wordt het bijhorende ticket immers automatisch verplaatst van *in development* naar *in code review*, dankzij de *pull request created-trigger.* Bovendien kan de ontwikkelaar bij het indienen van zijn *pull request* eveneens één of meerdere *reviewers* aanduiden. Deze *reviewers* krijgen dan meteen een melding in hun *Jira-inbox*, alsook een e-mail, waarin de *pull request* en het gerelateerde ticket worden aangegeven.




*figuur 42 - voorbeeld van een commit-message met Jira-ID via Sourcetree*

### Van in code review naar done

*Code reviewers* kunnen nu aan de slag gaan met de ontwikkelde code. Eens een reviewer tevreden is met het geleverde werk, kan hij in het corresponderende ticket het *pull request* om naar *development* te *mergen* goedkeuren. Eens een ticket door alle nodige *reviewers* goedgekeurd is, kan de *merge* plaatsvinden. Het is belangrijk dat de projectmanager de juiste restricties instelt, zodat niet

iedereen gemachtigd is om de merge uit te voeren. Bij voorkeur kan enkel een *senior developer* een dergelijke actie uitvoeren. De *merge* wordt dan eveneens van in het ticket in Jira uitgevoerd. Door de p*ull request merged-trigger* verplaatst Jira het ticket automatisch van *in code review* naar *done*.

#### Van done naar in UAT

Eens een *issue* als *done* gemarkeerd is, zijn de gerelateerde *features* klaar voor *deployment*. Hier volgen de acceptatietesten op. Deze testen gebeuren extern door de eindgebruiker, en worden dus niet geautomatiseerd, en vallen buiten de scope van dit onderdeel.

#### Van In UAT naar tested

Wanneer de *issues* getest zijn door de eindgebruiker kunnen ze verplaatst worden naar *tested*. Als een *feature* niet naar wens werkt of als er *bugs* gevonden zijn, kunnen de *issues* terug in de *backlog* opgenomen worden.

## 4.6 Verdere automatisatie met Automation

### 4.6.1 Inleiding

Automation is een plug-in voor Jira waarmee een heel aantal processen binnen Jira geautomatiseerd kunnen worden. Met behulp van die plug-in kunnen we in het kader van projectmanagement en met het oog op DevOps een heleboel automatisaties realiseren, waardoor de algemene projectkwaliteit zal verbeteren. Denk hierbij bijvoorbeeld aan volgende aspecten die we kunnen bekomen door automatisatie:

a) De betrouwbaarheid, volledigheid en traceerbaarheid van *issues* vergroten
b) De werklast van ontwikkelaars en projectmanagers reduceren
c) Problemen zo vroeg mogelijk detecteren
d) De snelheid van de development cyclus verhogen
e) Tevredenheid van de projectteams verhogen
f) Rapportage naar de klant toe verbeteren

Tal van automatisaties kunnen gerealiseerd worden met de gratis versie van de plug-in; Automation-lite. Wanneer de functionaliteit om e-mails en Slack-berichten te versturen ook wenselijk is, moet er overgeschakeld worden naar de betalende versie.

Automation is gebruiksvriendelijk, maar voor meer complexe automatisaties, is enig technisch inzicht en enige kennis van JQL vereist, maar daarover volgt later in deze paper meer.

Verder is een enorm voordeel ook dat automatisatieregels gekoppeld kunnen worden aan één of meerdere projecten, of gewoon globaal over alle projecten kunnen toegepast worden. Bovendien zijn dieregels ook eenvoudig te importeren en exporteren. Wanneer de projectmanager dus ingeschakeld wordt op consultancy-basis, kan hij zijn eigen regels toepassen in andere *Jira-environments*, zonder die telkens volledig opnieuw te moeten opstellen.

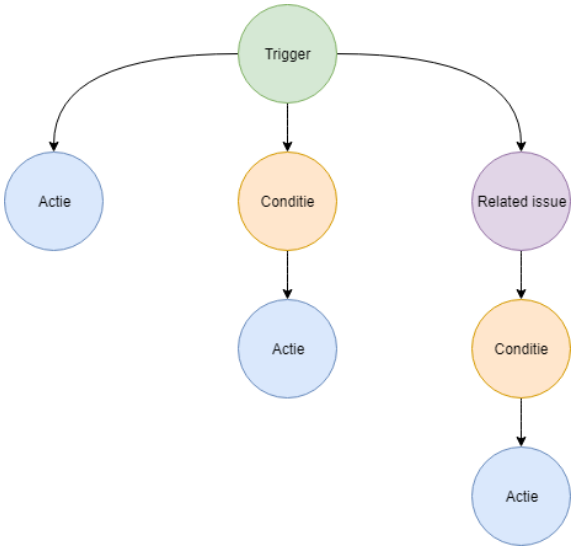
### 4.6.2 Werking van Automation

Elke automatisatie die gerealiseerd kan worden, wordt een *rule* of regel genoemd.

Een regel wordt telkens volgens een bepaalde structuur samengesteld.

Het is zo dat een regel telkens vanuit een *trigger* wordt aangeroepen. Tal van gebeurtenissen kunnen kwalificeren als *trigger*. Denk bijvoorbeeld aan:

a) Een tijdsinterval, bijvoorbeeld elke 10 minuten, wekelijks, elke maandagochtend, …
b) Een transitie, bijvoorbeeld een issue die overgaat van *backlog* naar *ready for dev*
c) Een *issue*-veld dat wordt aangepast, bijvoorbeeld het *description-field* dat wordt gewijzigd



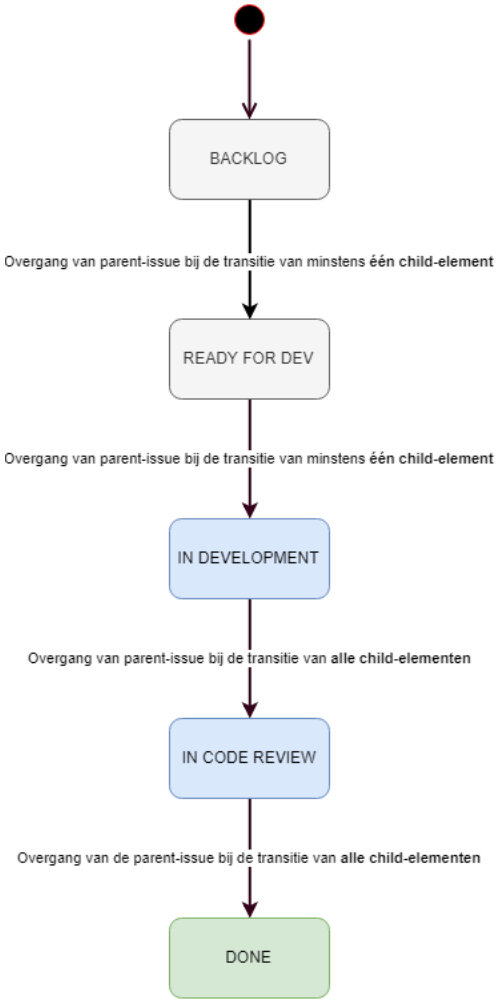

*figuur 43 - Schematische voorstelling voor het opstellen van een regel in Automation*

Vervolgens kan er meteen een actie volgen op de trigger. Zo kunnen we bijvoorbeeld wekelijks een e-mail uitsturen naar alle betrokken projectleden.

Anderzijds kunnen we ook een conditie een bepaalde actie laten voorafgaan. Een dergelijke conditie kan bijvoorbeeld een *JQL-query* zijn, waarbij er wordt gezocht naar *high-priority issues*. We kunnen ook eveneens meteen gaan kijken naar *related issues,* bijvoorbeeld *sub-tasks,* om de regel samen te stellen.

### 4.6.3 Transities van parent-elementen bij child-element transities

Binnen Jira is het vaak nuttig om binnen een issue te werken met subtaken. Zo kunnen bijvoorbeeld frontend- en backend-taken gekoppeld blijven aan het *parent issue*. Dit is ideaal wanneer er verschillende teams voor uiteenlopende doeleinden werk verrichten voor één *issue*.




*Figuur 44 - Workflow voor parent- en child-elementen*

Een probleem dat echter optreedt binnen Jira, is wanneer we subtaken, oftewel *child*-elementen, verplaatsen van de ene status naar de andere, zoals eerder omschreven. We verwachten dat, wanneer een *child*-element bijvoorbeeld van de *backlog* naar *ready for dev* verplaatst wordt, dat het *parent*-element dan automatisch mee zou verplaatst worden. In de praktijk is dit echter niet zo. Sterker nog, het is niet mogelijk om deze werking te realiseren binnen Jira, zonder daar een externe plug-in voor te implementeren en configureren. Dit is echter een probleem wanneer we willen werken met automatisatie binnen de workflow van Jira. Het doel is immers om de ontwikkelaars zoveel mogelijk te ontlasten, door de tijd die ze moeten investeren in het up-to-date houden van Jira tot het absolute minimum te reduceren en eveneens de betrouwbaarheid van automatisch gegenereerde voortgangsrapporten binnen de Atlassian toolkit te vergroten.  Daarom is het belangrijk om de overgang van parent-elementen met hun child-elementen ook te automatiseren.

Vooraleer een plug-in geïnstalleerd kan worden, is het belangrijk na te denken over een aantal regels die steeds gevolgd moeten worden om de correcte flow van de betrokken elementen te garanderen.

We kunnen hier de volgende regels vastleggen, zoals voorgesteld in bovenstaande figuur:

- Een *parent*-element moet van *backlog* naar *ready for dev* verplaatst worden bij de overgang van minstens één *child*-element
- Een *parent*-element moet van *ready for dev* naar *in development* verplaatst worden bij de overgang van minstens één *child*-element
- Een *parent*-element mag uitsluitend van *in development* naar *in code review* verplaatst worden bij de overgang van *alle* verbonden *child*-elementen.
- Een *parent*-element mag uitsluitend van *in code review* naar *done* verplaatst worden bij de overgang van *alle* verbonden *child*-elementen.

Elk van bovenstaande transities, moet individueel opgesteld worden. Elke transitie van een subtaak kan immers een *trigger* zijn voor de transitie van de hoofdtaak.

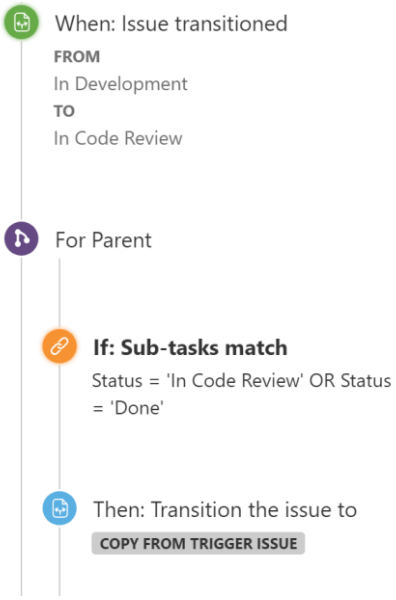
We krijgen dus volgende regels (zoals voorgesteld in de afbeeldingen):


Deze regel geeft aan dat, wanneer een sub-task van *backlog* naar *selected for development* wordt verplaatst, dat de *parent issue* eveneens wordt verplaatst naar *selected for development.*

Dezelfde structuur wordt eveneens behouden voor *issues* die van *selected for development* naar *in development* worden verplaatst. Hier kunnen we eveneens aan koppelen dat *issues* toegekend worden aan de ontwikkelaar die het *issue* in deze status plaatst. Deze persoon zal bovendien het issue *in development* plaatsen, door het aanmaken van een *feature-branch*, zoals al eerder gerealiseerd is dankzij *workflow triggers*. Voor een concrete invulling van deze regel, zie 4.6.4.

*figuur 45 - Transitie van parent bij de overgang van een child uit backlog*

Voor statussen waar de regel is dat alle sub-taken moeten overgegaan zijn naar deze status, vooraleer de *parent* mag overgaan, kunnen we een regel zoals hieronder opstellen:




*figuur 46 - regel voor parent-transitie indien alle children voldoen aan een conditie*

### 4.6.4 Het automatisch toekennen van issues aan een ontwikkelaar

Het is uiteraard de bedoeling dat een *issue* zo vroeg mogelijk in de *workflow* toegekend wordt aan de ontwikkelaar die ermee aan de slag zal gaan, zodat de projectmanager steeds een goed overzicht heeft, en zodat een realistische verdeling van de *workload balance* kan opgenomen worden in de sprintplanning en de planning van resources. In principe gebeurt deze toekenning dus al manueel

wanneer een *issue* opgenomen is in de huidige sprint. Deze functie moet dus vooral als voorzorgsmaatregel beschouwd worden.

We stellen deze functie op volgens volgende regels:

a) De ontwikkelaar die een *branch* creëert, wordt automatisch de *assignee* van het ticket in kwestie. Dit wordt eveneens mogelijk gemaakt door de *workflow*-automatisatie zoals eerder besproken.
b) Deze automatisatie wordt enkel getriggerd als er nog geen *assignee* voor het corresponderende ticket is.
c) Als een ticket via deze weg toegekend wordt, dan wordt de *assignee* hiervan op de hoogte gebracht via notificatie.
d) Als het ticket in kwestie een *child*-element is; bijvoorbeeld een subtaak gekoppeld aan een *parent*-element, dan wordt er eveneens een check uitgevoerd voor het *parent*-element. Indien het *parent*-element ook geen *assignee* heeft, dan wordt de persoon die de *branch* aanmaakt voor het *child*-element, ook de *assignee* van het *parent*-element.

Merk ook op dat het toekennen van een *parent issue* op deze wijze een voorzorgsmaatregel is, om te voorkomen dat *issues* niet toegekend worden door een menselijke fout, en daarom ook moeilijk traceerbaar worden voor de projectmanager. Vandaar dat, als een toekenning op deze manier gebeurt voor een parent-issue, dat de projectmanagers dan verwittigd worden via e-mail. We kunnen dan volgende boodschap instellen, met behulp van *regular expressions.*

***Subject:*** *Parent issue {{issue.key}} has automatically been assigned*

***Content****:*

*Hello!*

*Just letting you know:*

*Issue {{issue.key}} is a parent issue with subtasks that had no assignee.*
*{{initiator.displayName}} has been automatically assigned because this issue or a subtask has been moved to [In development].*

*Please take a look at the issue here:*

*{{issue.toUrl}}*

Een mogelijke voorbeeld output via e-mail is dan:

***Subject**: Parent issue JIR-15 has automatically been assigned*

***Content**:*

*Hello!*

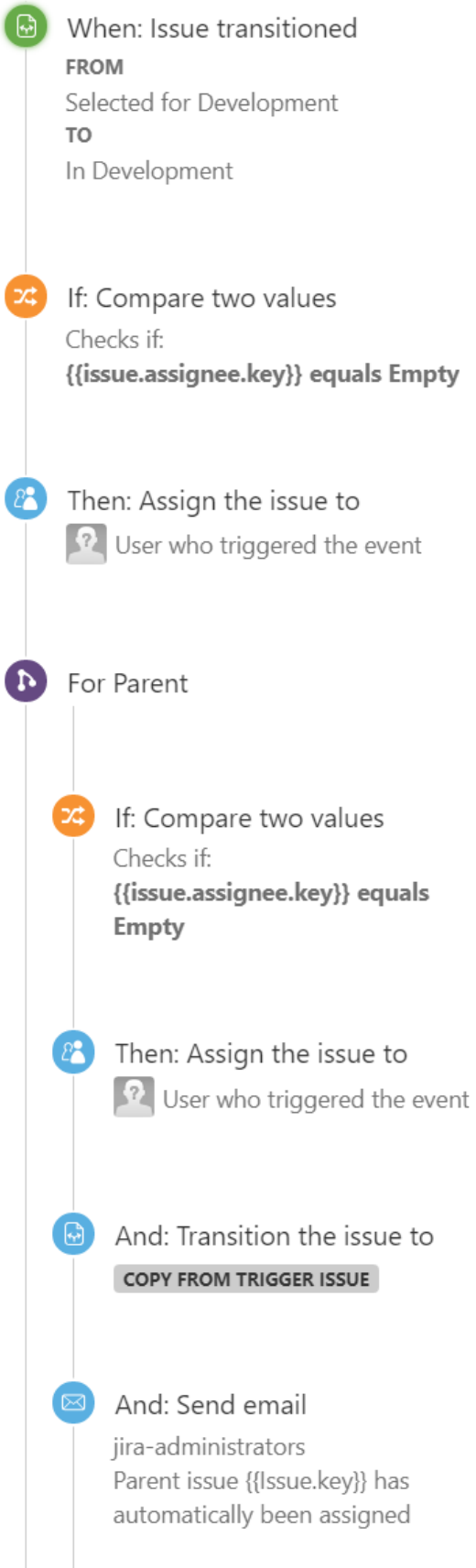
*Just letting you know:*

*Issue JIR-15 is a parent issue with subtasks that had no assignee. Stephanie Govaers has been automatically assigned because this issue or a subtask has been moved to [In development].*

*Please take a look at the issue here:*

*https://companyname.atlassian.net/browse/JIR-15*

Deze complexere regel ziet er dan als volgt uit:




*figuur 47 - Regel voor het automatisch toekennen van issues*

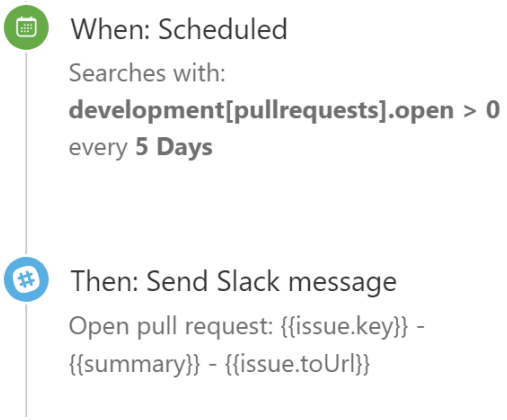
Een belangrijke opmerking bij deze automatisatie is dat er ook moet nagedacht worden of er nog van de 'ideale', automatische *workflow* kan of mag afgeweken worden. Als ervan uitgegaan wordt dat een *issue* uitsluitend automatisch van *ready for development* naar *in development* verplaatst mag worden, dan moet er enkel een regel opgesteld voor dit scenario. Als er echter verwacht wordt dat *issues* ook manueel versleept moeten kunnen worden van bijvoorbeeld *backlog* naar *in development,* en daarbij de stap *ready for dev* overgeslagen wordt, dan moet voor dit scenario ook een regel omschreven worden.

### 4.6.5 Het verwittigen van een senior developer of projectmanager indien er openstaande pull requests zijn.

Het is erg belangrijk om openstaande *pull requests* zo goed mogelijk op te volgen. Momenteel kunnen openstaande *pull requests* opgevolgd worden binnen Bitbucket. Het nadeel is hier dat de *pull requests* daar enkel binnen een individuele *repository* zichtbaar zijn. Dit betekent dat het overzicht verloren gaat wanneer de gebruiker betrokken is in meerdere projecten. *Pull requests* kunnen eveneens opgevolgd worden in Jira, zowel project-specifiek als globaal. Het probleem hier is dan echter dat, om een volledig overzicht te bekomen, projectmanager telkens handmatig een *JQL-query* zal moeten uitvoeren om de openstaande *pull requests* te zien, om vervolgens zelf de betrokken *developers* op de hoogte te brengen.

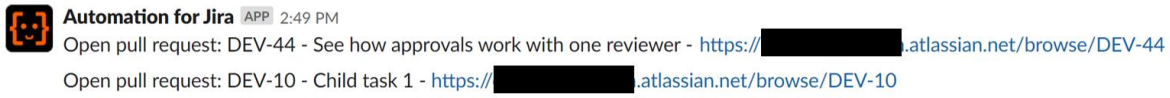
Om dit te voorkomen, stellen we een regel in die op vaste tijdstippen de betrokken personen op de hoogte brengt van openstaande *pull requests*. Op deze manier kunnen we bijvoorbeeld garanderen dat *code reviews* niet vergeten worden, of dat *pull requests* te laat worden *gemerged*.

Onze regel ziet er dan als volgt uit:




*figuur 48 - regel om gebruikers op de hoogte te brengen van openstaande pull requests*

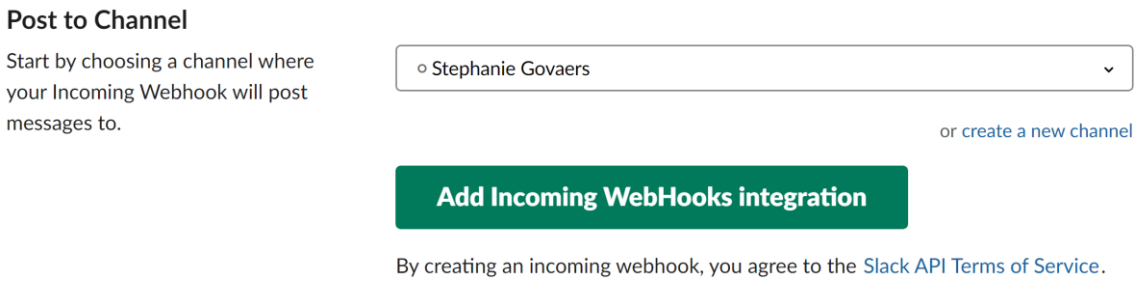
We krijgen dan bijvoorbeeld onderstaande *messages* in ons kanaal:



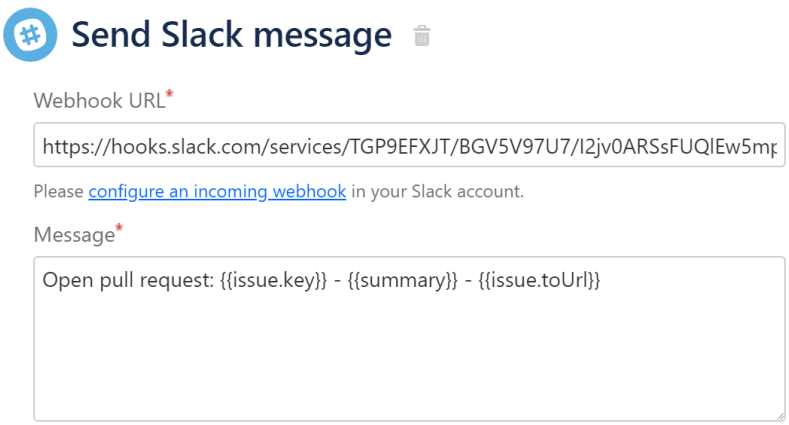

*figuur 49 - Automation messages in Slack*

Een belangrijke opmerking is hier dat er uit noodzaak is gekozen voor het versturen van Slack-berichten. Bij voorkeur zou er op vaste tijdsintervallen één e-mail verstuurd worden waarin de betrokken issues opgesomd worden. Zo is het bijvoorbeeld dat, in de *server*-versie van Automation, alle *issues* die voortkomen uit een *JQL-query* opgesomd kunnen worden in een e-mail met behulp van *Hypertext Markup Language (*HTML). Helaas ondersteunt de *cloud*-versie van Automation deze handeling niet. In de *cloud*-versie wordt er een aparte e-mail voor elk opgeleverd *issue* verstuurd. Stel dat een gebruiker gelinkt is aan vijftien *issues*, dan kan hij zich ook aan vijftien e-mails verwachten. Dit is uiteraard niet wenselijk en zelfs contraproductief. De berichten op Slack worden op dezelfde manier verstuurd, maar daar heeft de ontvanger uiteraard minder last van, aangezien alle berichten tegelijk aankomen in één kanaal.

Een groot nadeel bij het versturen van dergelijke berichten via Slack is echter dat, wanneer een bericht moet verstuurd worden naar een gebruiker, enkel deze gebruiker de ontvanger mag zijn. Om dit te realiseren, moet een *WebHook* geconfigureerd worden voor *elke* gebruiker. Bovendien wil dit ook zeggen dat er voor iedere gebruiker een aparte regel moet opgesteld worden. Deze werking is dus alleen toepasbaar voor relatief kleine organisaties met kleine teams.




*figuur 50 - het genereren van een WebHook integratie, wat zal resulteren in een JSON URL*



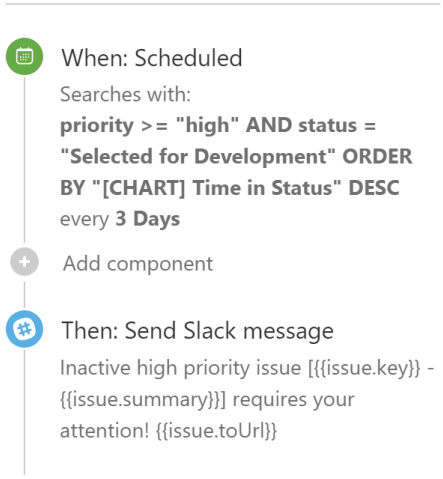

*figuur 51 - het toevoegen van een WebHook*

### 4.6.6 Projectmanager en toegekende jira-gebruikers notificeren van openstaande high-priority issues via Slack.

Het is doorgaans de bedoeling dat issues met een hoge prioriteit het eerste afgehandeld worden, en dus niet in *selected for development* blijven staan. In de praktijk kan er mogelijk over een *issue* met een hoge prioriteit gekeken worden, en wordt er misschien aan *issues* gewerkt die een lagere prioriteit hebben. Daarom is het erg belangrijk dat hier ook een controlemiddel is. Op deze manier kunnen problemen zoals *bottlenecks* tijdens het *development-proces* immers vermeden worden.

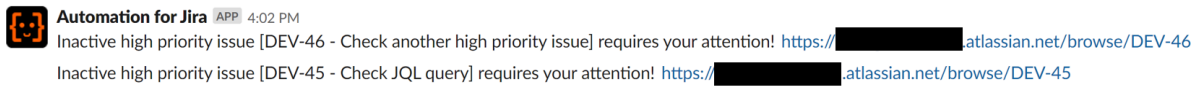
Zoals in 3.6.5 omschreven, is er vanuit dezelfde overwegingen gekozen om te werken met Slack-berichten en niet met e-mails.

Om deze berichten om de drie dagen, en geordend te tonen, kunnen we onze regel als volgt samenstellen:




*figuur 52 - Regel om gebruikers op de hoogte te brengen van openstaande issues met een hoge prioriteit*

Deze regel resulteert in volgende voorbeeldoutput:

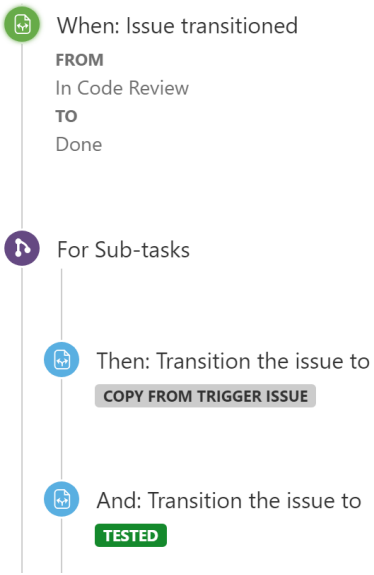



*figuur 53 - voorbeeldoutput in Slack*

### 4.6.7 Alle subtaken van één parent-issue van done naar tested verplaatsen.

Tijdens het ontwikkelproces is het zinvol om met subtaken gekoppeld aan een hoofd-*feature* te werken. Op deze manier kunnen we bijvoorbeeld frontend- en backend-taken van elkaar gescheiden houden. Eens alle gerelateerde taken echter als *done* gemarkeerd staan, en zouden moeten overgaan naar de UAT-fase, zijn de subtaken niet meer van belang. Enkel de hoofdtaak zal immers getest worden, en niet elke individuele *sub-task*. Via automatisatie kan er gerealiseerd worden dat eens een *parent issue* als *done* gemarkeerd is, alle sub-taken naar *tested* verplaatst worden. Op deze manier zijn de subtaken nog steeds traceerbaar, maar is het overzicht van tickets die klaar zijn voor UAT veel duidelijker, en is het kanbanbord niet overvol. Bovendien kan er eventueel ook gewerkt worden met onzichtbare statussen, waardoor de subtaken niet meer zichtbaar zijn voor andere gebruikers dan de projectmanager.

Volgende regel maakt deze wens waar:




*figuur 54 - regel om subtaken naar tested te verplaatsen*

### 4.6.8 Besluit

Naast de enorme praktische voordelen van het automatisatieproces op zich, genereert de plug-in ook *audit logs* van de verschillende automatisaties op regelmatige intervallen. Deze logs zijn terug te vinden per automatisatieregel. Bovendien wordt de *project administrator* ook op de hoogte gebracht van falende automatisaties en conflicten via e-mail. Deze waarschuwingen zijn bovendien ook duidelijk zichtbaar in het *dashboard* van de plug-in.

Er zijn nog tal van andere automatisaties mogelijk, maar merk op dat hier rekening gehouden moet worden met *GDPR-compliancy.* Wanneer we dus gebruiker-specifieke automatisatie willen verwezenlijken, mogen we nooit verwijzen naar *usernames*. In de plaats daarvan moeten we gebruik maken van het *Jira-account-id* van een gewenste gebruiker. Gelukkig voorziet de plug-in zelf een automatische omzetting van *username* naar ID. Dit is ideaal wanneer we zelf een *query* willen schrijven voor een specifieke *user*. Het is echter wel problematisch wanneer we een output in Jira willen creëren met een *username*. Deze zal dan immers ook in de output omgezet worden naar een ID, waardoor de output niet meer leesbaar is.

## 4.7 Continuous integration (CI) met Bitbucket

### 4.7.1 Wat is Continuous integration?

Het testen van software is een cruciaal onderdeel van *software development*. Bovendien kan aan het testen van software een erg hoog kostenplaatje verbonden zijn. Er moet bovendien niet enkel getest worden of nieuwe *features* werken naar behoren, maar eveneens dat bestaande *features* nog blijven werken. Hoe meer *features* er ontwikkeld worden, hoe meer *features* er getest moeten worden. Dit betekent dat bij elke *release* de *scope* van testen vergroot, waardoor de takenlast en kostprijs enorm snel kan oplopen. Als deze testen dan ook nog manueel uitgevoerd worden, is de uitvoering ervan bijna onmogelijk. Een eerste stap is dus het automatiseren van de testen.

Er kan echter nog een stap verder gegaan worden. Door gebruik te maken van Bitbucket Pipelines, kunnen testen automatisch uitgevoerd worden voor elke *change* die wordt *gepusht*. [13]

Op deze manier kan *continous integration (CI)* toegepast worden. CI is werkwijze waarbij alle ontwikkelaars aan één gemeenschappelijke *repository* werken. Zoals al eerder omschreven, werken de ontwikkelaars op een *feature-branch*, gelinkt aan aan Jira-ticket. Wanneer een *developer* klaar is met een bepaalde *feature*, kan hij een *pull request* uitvoeren om te *mergen* naar de *develop branch*. In het geval van CI worden elk van deze *check-ins* geverifieerd door een automatische *build,* inclusief testen. [14] Aangezien code voortdurend “geïntegreerd” wordt door alle ontwikkelaars, wordt er gesproken van CI.
In dit geval zal Bitbucket Pipelines als *Continuous Integration Service* gebruikt worden

### 4.7.2 De voordelen van continuous integration

Door het toepassen van CI, kunnen ontwikkelaars meer tijd spenderen aan het ontwikkelen. Het is immers zo dat problemen eenvoudiger zijn om te voorkomen, en mochten ze dan toch voorkomen, is het veel eenvoudiger om te traceren waar het precies is misgelopen.

Kosten kunnen eveneens gedrukt worden door CI. De tijd tussen verschillende integraties is immers aanzienlijk kleiner, waardoor problemen eveneens sneller gevonden en opgelost kunnen worden. Op deze manier kan er dus voorkomen worden dat deadlines niet gehaald worden, of dat projecten zelfs in hun geheel falen. Software kan immers sneller opgeleverd worden, aangezien de *repository* altijd in een *deployable state* is.

Het is een weloverwogen beslissing om binnen dit onderzoek te werken met Bitbucket Pipelines.

### 4.7.3 Werking in het projectteam

De implementatie van een *CI-service* is uiteraard al een stap in de juiste richting. Toch is CI enkel werkbaar wanneer het projectteam ook de bijhorende *mindset* adopteert.

Het is belangrijk dat *developer*s regelmatig hun code inchecken. Daarnaast mogen geen *pull requests* gebeuren voor *broken code.* Mocht dit toch gebeuren, dan mag de *merge* al zeker niet uitgevoerd worden. Bitbucket voorziet voldoende waarschuwingen, maar menselijke handelingen kunnen die waarschuwingen nog steeds omzeilen. Het is dus ook belangrijk om als teamspeler de nodige verantwoordelijkheid op te nemen. Bovendien krijgen *developers* ook de verantwoordelijkheid om voldoende tijd en zorg in hun unit tests te steken.

### 4.7.4 Opzetten van Bitbucket Pipelines

In het kader van dit onderzoek, is een implementatie gebeurd aan de hand van een Java-project waarbij gebruik gemaakt wordt van Maven.

Dit voorbeeldproject bevat enerzijds een hoofdklasse met enkele simpele methodes zoals op onderstaande afbeelding getoond.

```
public class MainApp {


    public int add(int arg1, int arg2) {
        return arg1 + arg2;
    }

    public int subtract(int arg1, int arg2) {
        return arg1 - arg2;
    }

    public int multiply(int arg1, int arg2) {
        return arg1 * arg2;
    }

    public int divide(int arg1, int arg2) {
        return (arg2 == 0) ? 0 : arg1 / arg2;
    }


}
```

*figuur 55 - voorbeeld java-klasse*

En anderzijds een testklasse waarin de functionaliteit van de methodes wordt getest zoals hieronder getoond.

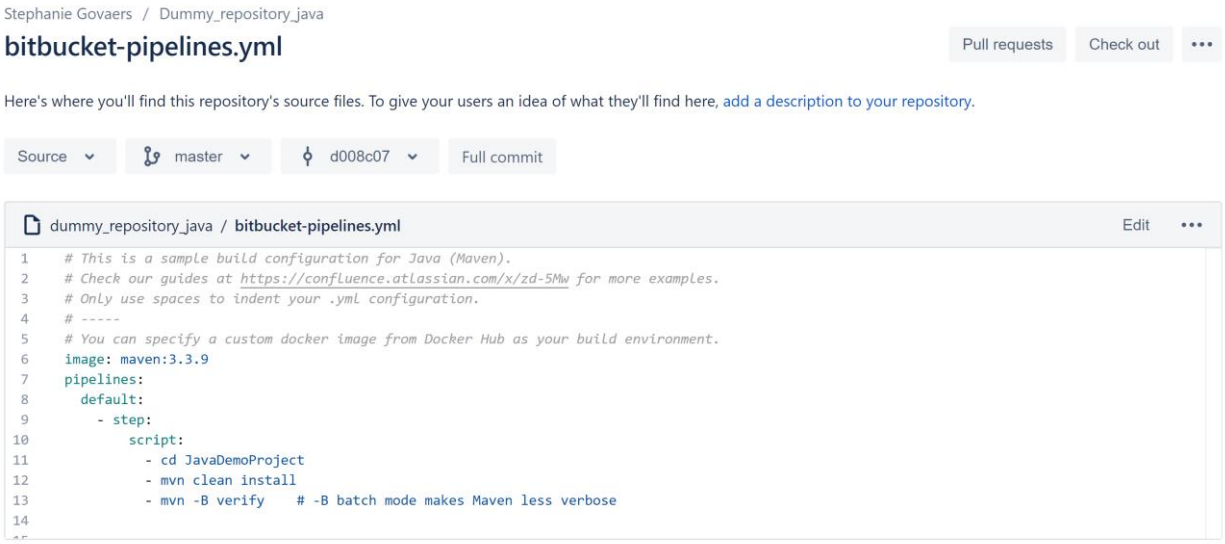
```
    @Test
    public void testAdd() { assertEquals(test.add(1, 1), actual: 2); }

    @Test
    public void testSubtract() { assertEquals(test.subtract(1, 1), actual: 0); }

    @Test
    public void testMultiply() { assertEquals(test.multiply(1, 1), actual: 1); }

    @Test
    public void testDivide() { assertEquals(test.divide(1, 1), actual: 1); }
}
```

*figuur 56 - voorbeeldetesten Java-project*

Deze code wordt opgeslaan in een *repository* op Bitbucket en wordt zo eveneens aan Jira gekoppeld, zoals al in voorgaande hoofdstukken omschreven. Op deze manier kunnen we ook weer de verschillende automatisaties gaan toepassen.

Om een *pipeline* op te zetten, vertrekken we vanuit de *repository* op Bitbucket. Via Bitbucket Pipelines moet er vervolgens een script gegenereerd worden waardoor de *pipeline* in werking zal treden bij elke *pull request* om te *mergen.* Dit script wordt vervolgens aan de *repository* toegevoegd onder de naam **bitbucket-pipelines.yml.**

In dit geval ziet het script er dan als volgt uit:




*figuur 57 - Bitbucket Pipelines script voor een Maven project*

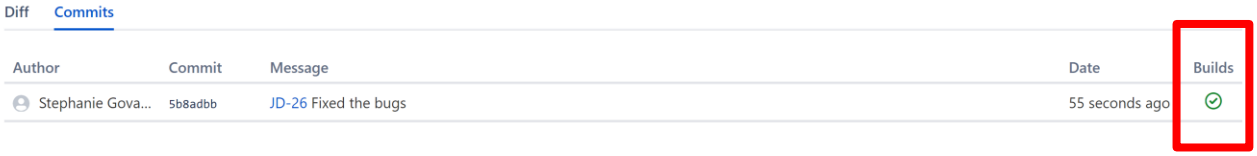
Bij elke *pull request* om een *merge* naar de *development branch* uit te voeren, wordt het script uitgevoerd. Merk op dat in dit script het commando **cd JavaDemoProject** wordt uitgevoerd, waarbij 'JavaDemoProject' de naam van onze *root folder* is. Dit is noodzakelijk in elk Maven-project, aangezien Bitbucket Pipelines automatisch naar de *Project Object Model (**POM**)-file* zal zoeken in volgende mappenstructuur binnen de *repository*:

*/opt/atlassian/pipelines/agent/build.*

Deze mappenstructuur wordt niet automatisch gegenereerd. Maven daarentegen zal de *POM-file* automatisch in de *root* van de repository zetten. Vandaar dat er in dit script ook naar die locatie genavigeerd wordt.

Daarna zal er telkens een *clean install* van Maven uitgevoerd worden. Vervolgens wordt de *repository gebuild,* en worden de *unit tests* uitgevoerd ten opzichte van de *development branch*.

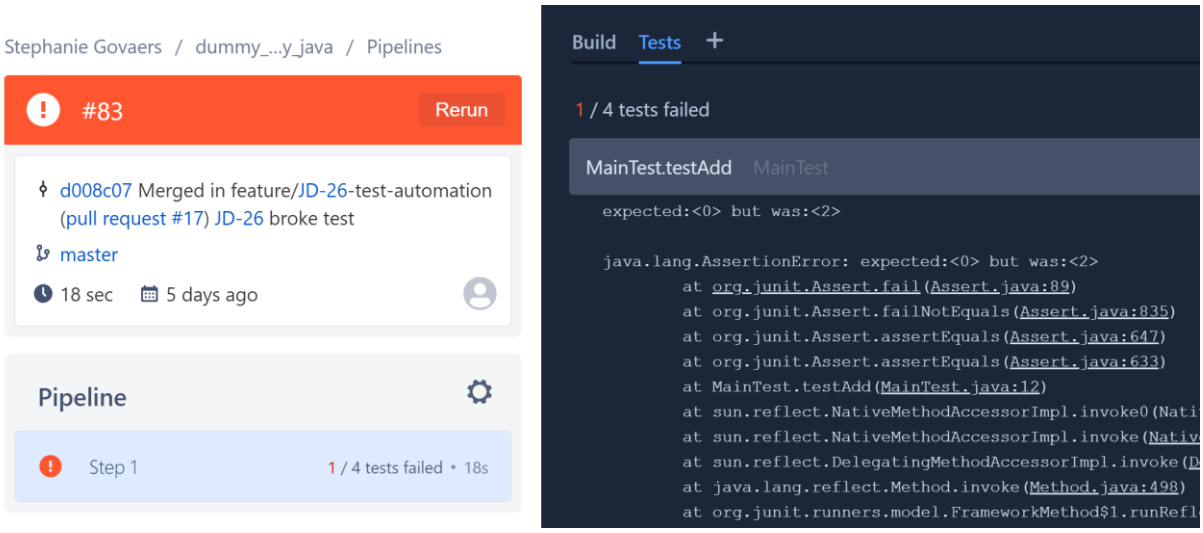
Bitbucket geeft dan onmiddellijk feedback over het al dan niet slagen van het *build*-proces en de unittesten. Dit wordt grafisch weergegeven op bij het indienen van een *pull request* binnen Jira zoals hieronder afgebeeld.




*figuur 58 - Pull request on web met geslaagde build en unit tests*

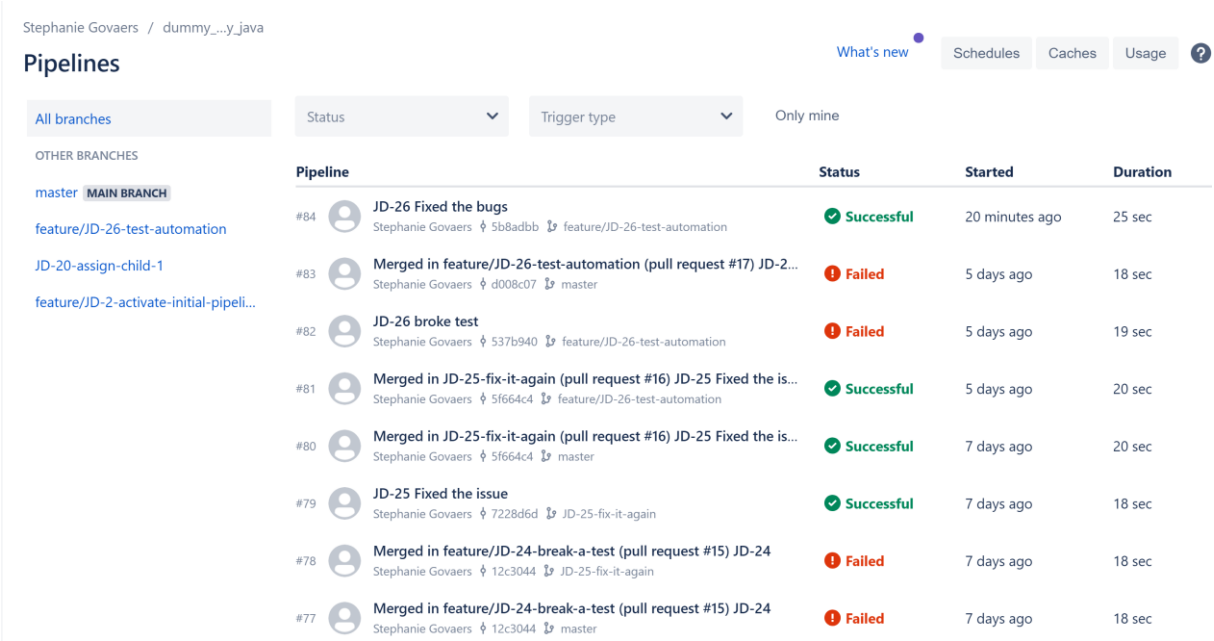
Indien de *build* en *unit tests* slagen, wordt een groen vinkje getoond. Zo niet, wordt een rood kruisje getoond. Bovendien kunnen we binnen Bitbucket ook meteen nagaan waar het precies is misgelopen. We krijgen dan het scherm zoals afgebeeld op figuur 59. Als er een *unit test* gefaald is, geeft Bitbucket eveneens aan welke test en met welke resultaten in de vorm van een *stack trace.*

We kunnen op dit punt zeggen dat we CI gerealiseerd hebben. Zoals al eerder aangekaart, is verantwoordelijkheid van het team hier natuurlijk cruciaal. *Unit tests* moeten met voldoende zorg geschreven worden, en wanneer Bitbucket aangeeft dat de code niet werkt, mag de *pull request* zeker niet uitgevoerd en uiteindelijk *gemerged* worden.




*figuur 59 - Failing build in Bitbucket Pipelines*

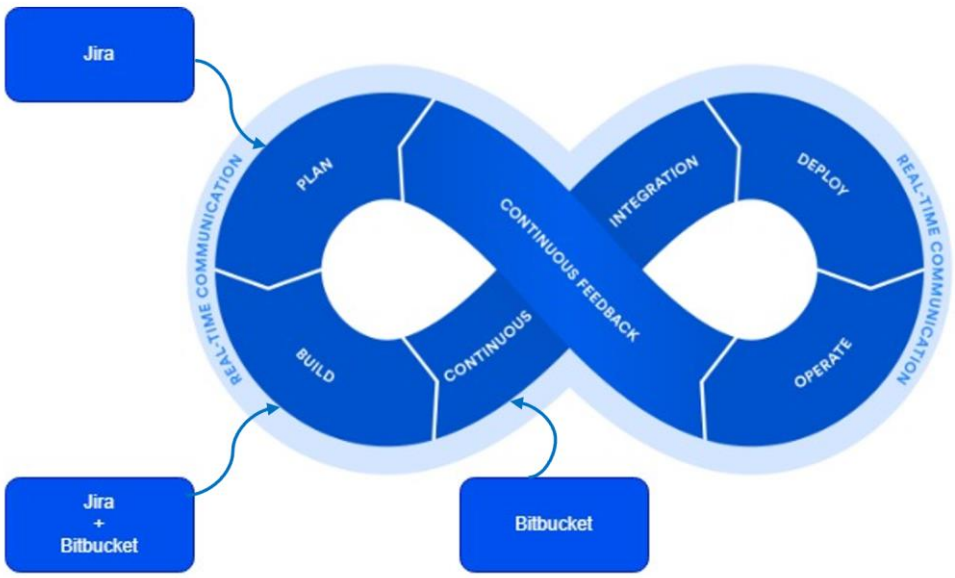
Tot slot is er ook nog steeds het overzicht van elke keer dat de CI getriggerd wordt in Bitbucket Pipelines. Alle geslaagde en falende *builds* worden hier duidelijk weergegeven, waardoor de traceerbaarheid van mogelijke problemen ook erg groot is.




*figuur 60 - overzicht van alle pipelines*

### 4.7.5 Overzicht Atlassian pipeline

De implementaties zoals omschreven in voorgaande hoofdstukken resulteren in de *software delivery pipeline* zoals afgebeeld op bovenstaande figuur. Door het toepassen van de automatisatie in de *workflow* en via de plug-in met de combinatie van CI, kunnen we hier al spreken van een *DevOps pipeline*. We kunnen hier nog niet spreken van een *end-to-end pipeline,* aangezien de focus nog voornamelijk ligt op de eerste fasen van het *software development proces*.




*figuur 61 - Atlassian DevOps pipeline*

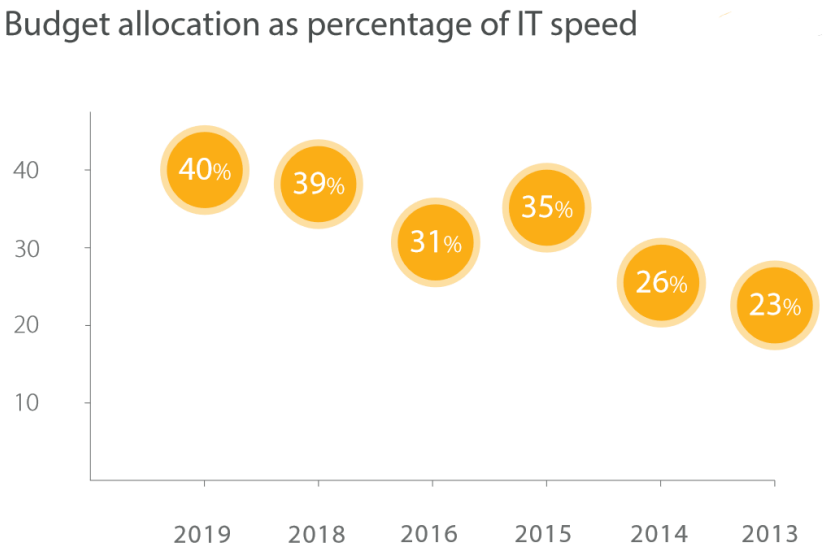
## 4.8 Overzicht toegepaste DevOps-principes

| Omschrijving principe | | Verklaring |
|---|---|---|
| Accountability, ownership & end-to-end responsibility | ✔ | Door de traceerbaarheid, volledigheid en correctheid van werk te vergroten dankzij een betere invulling van Jira-issues, optimalisatie van Jira-workflows en geautomatiseerde reminders voor bepaalde taken met *Automation for Jira*. |
| Collaboration | ✔ | Door de traceerbaarheid, volledigheid en correctheid van werk te vergroten dankzij een betere invulling van Jira-issues, hebben teams een betere basis voor communicatie en samenwerking. Problemen kunnen immers sneller gedetecteerd worden door deze verbetering, met als gevolg dat het team sneller ondersteuning kan bieden. |
| Automation | ✔ | Automatisatie is gerealiseerd door regels op te zetten binnen de workflows van Jira, maar ook door bijkomende automatisering waar te maken door het gebruik van *Automation for Jira.* Hierdoor is het mogelijk gemaakt dat Jira-issues automatisch transitioneren, Jira-issues automatisch worden toegekend aan de juiste persoon, en worden automatisch mails of Slack-berichten verstuurd bij openstaande *high-priority issues* of openstaande *pull requests.* |
| Continuous Integration | ✔ | Continuous Integration is gerealiseerd door de code te linken aan een *repository* op Bitbucket, waarbij vervolgens de *unit tests* zullen uitgevoerd worden bij elke *build*. |
| Continuous Improvement | ✔ | Bovenstaande realisaties kunnen allemaal in het teken staan van Continuous Improvement. De omschreven verbeteringen kunnen immers de taaklast van de *developers* verminderen en de efficiëntie van de projectmanagers vergroten. Naast deze interne verbeteringen, kunnen deze aanpassingen ook naar externen een enorme verbetering zijn. Denk bijvoorbeeld aan de verbeterde gegenereerde rapporten in Jira, door de toegenomen betrouwbaarheid van de informatie op Jira. |
| Fast & reliable deploys with Continuous Delivery | ✗ | Zoals eerder omschreven valt CD buiten de scope van dit project. Het is echter zo dat bovenstaande DevOps-realisaties een ideale basis zijn om CD op een optimale manier te kunnen implementeren. |

# 5 Kwaliteit door Test Management en Behavior-Driven Development in een DevOps-setting

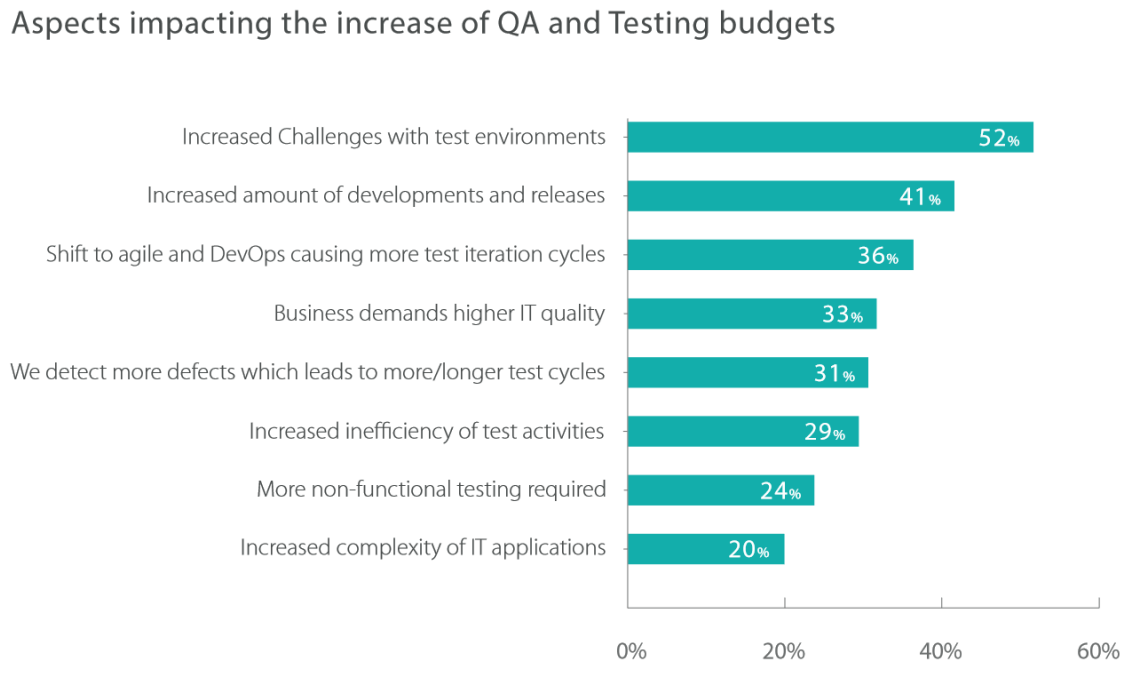
## 5.1 Inleiding

*Quality Assurance* wordt alsmaar belangrijker in IT-projecten. Sinds 2013 wordt steeds meer en meer budget uitgegeven aan QA. In 2019 wordt zelfs 40% van het totale budget voor een IT-project toegekend aan QA. [15]




*figuur 62 - Budgetten toegekend aan QA*

Zoals afgebeeld op de grafiek hieronder, zijn er heel wat aspecten binnen een IT-project die de toenemende kostprijs voor QA rechtvaardigen.




*figuur 63 - oorzaken van de toenemende kostprijs van QA*

Vanwege het grote belang van testen in IT-projecten, is er in dit hoofdstuk onderzoek gevoerd naar een waardevolle aanvulling voor een *DevOps-pipeline* en verdere Jira-integratie. Hiervoor is gebruik gemaakt van de Xray-plug-in voor Jira, waarmee rechtstreeks in Jira aan *test management* gedaan kan worden en waarbij testen eveneens opgenomen kunnen worden in de *CI-pipeline* aan de hand van testautomatisatie. Bovendien is er ook uitvoerig onderzoek gedaan naar *Behavior-Driven Development* (BDD) om ook de juiste methodieken toe te passen om op een waardevolle manier aan *test management* te doen en een aantal andere probleemstellingen te kunnen beantwoorden.

## 5.2 Doelstellingen

Door DevOps-principes toe te passen op *test management* en door *Behavior-Driven Development*, kunnen we de algemene projectkwaliteit verbeteren. Bovendien kan er op deze manier ook voor gezorgd worden dat projectmanagers hun taken beter kunnen volbrengen. We kunnen door aan *test management* te doen binnen Jira concreet volgende doelstellingen verwezenlijken:

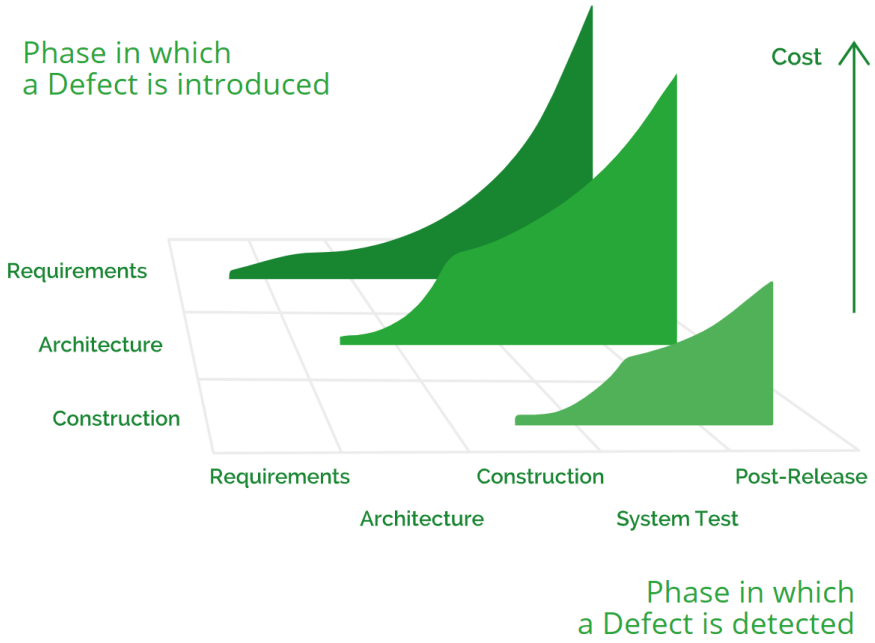
a) De kwaliteit van de opgeleverde software verhogen
b) De *time to ship* verkleinen door *builds* sneller te valideren
c) De bestaande *continuous integration* verbeteren
d) Kostprijs te drukken door testen te automatiseren en dus minder menselijke *resources* nodig te hebben en tegelijk *bugs* sneller te detecteren.
e) Menselijke fouten zoveel mogelijk beperken
f) Waardevolle testrapporten genereren
g) De stand van zaken beter begrijpen, opvolgen en ingrijpen waar nodig
h) De kloof tussen business en development dichten
i) Samenwerking, zoals kenmerkend voor DevOps, bevorderen
j) Changes opvolgen en accountability en traceability aanmoedigen.

## 5.3 Testautomatisatie

### 5.3.1 Inleiding

Zoals eerder omschreven, is QA ontzettend belangrijk en ook ontzettend duur binnen de *software development lifecyle.* Een onderzoek van NASA in 2004 heeft immers aangetoond dat de kostprijs van *bug fixing* exponentieel toeneemt, hoe eerder (in de *development cycle*) *bugs* in de software geïntroduceerd worden. Bovendien neemt de kostprijs eveneens exponentieel toe, hoe later in de *development cycle* de *bugs* ontdekt worden. [16]

Het is dus belangrijk dat het testproces al in de eerste ontwikkelingsstadia gestart wordt; zelfs al bij het specificeren van de *requirements*. Het is dus een goede praktijk om te werken aan de hand van *Test-Driven Development* (TDD), maar zoals in een volgend hoofdstuk wordt omschreven, kan er zelfs verder gegaan worden, en kan er zelfs gewerkt worden met BDD.




*figuur 64 -Link tussen kosten, introduceren van een bug en ontdekken van een bug*

### 5.3.2 Soorten testen voor testautomatisatie

#### 5.3.2.1 Unit tests

*Unit tests* kunnen snel en eenvoudig geïmplementeerd en onderhouden worden. *Unit tests* zijn dus een goede plaats om te beginnen met testautomatisatie. Deze testen evalueren in feite of het systeem juist gebouwd wordt en kunnen tegelijkertijd gebruikt worden om na te gaan of de *code coverage* naar wens is.

Elke taal of platform heeft eigen (aangeraden) *testing frameworks,* zoals JUnit voor Java en NUnit voor C#.

#### 5.3.2.2 Integration and component tests

Integratietesten of componenttesten evalueren of onderdelen van het volledige systeem naar behoren met elkaar en met het geheel integreren. Dergelijke testen zijn cruciaal om problemen te voorkomen naarmate het ontwikkelingsproces vordert.

#### 5.3.2.3 Service tests

*Service tests* of functionele testen, zijn testen die de businesslogica testen. Hierbij komt geen *User Interface* (UI) aan te pas. Deze testen zijn een zeer goede aanvulling op *unit tests* en kunnen eveneens opgenomen worden in de *continuous integration pipeline.* Deze testen passen ook perfect binnen het concept van BDD, en dus kunnen *frameworks* zoals Cucumber of RSpec gebruikt worden. [17]

#### 5.3.2.4 UI tests

*UI tests*, of *end-to-end tests* evalueren eveneens de eisen van klanten of van business, maar doen dit aan de hand van echte situaties, namelijk met de UI. Een typisch *framework* dat hiervoor gebruikt wordt, is Selenium.

*UI tests* zijn vaak aantrekkelijk en erg gewild door klanten, maar toch zijn er heel wat nadelen. Zo zijn *UI tests* vaak duur en geven ze zelden uitsluitsel over de oorzaak van mogelijke problemen. Het is vaak tijdsrovend om deze testen uit te voeren, en het is tot slot vrij moeilijk om deze testen op te nemen in de *CI-pipeline.* [17]

#### 5.3.2.5 Acceptance tests

Tot slot zijn er acceptatietesten. Deze testen zijn een manier om uitdrukkelijk na te gaan of de opgelegde specificaties gehaald zijn. [17] Deze testen passen ook perfect binnen BDD, en kunnen zeer eenvoudig geschreven en uitgevoerd worden met Cucumber, zoals in een komend hoofdstuk besproken zal worden.

## 5.4 Behavior-Driven Development

### 5.4.1 Probleemstelling

Een probleem dat kenmerkend is voor IT-projecten is dat er heel wat informatie verloren gaat tussen elke schakel in de communicatiestroom, zoals hieronder afgebeeld.


*figuur 65 - Bronnen van communicatie binnen een project*

Wanneer een business analist voor het eerst gaat samenzitten met de klant, kan er al heel wat informatie verloren gaan. Zo kan een klant bijvoorbeeld zelf niet goed weten hoe hij naar de business analist toe moet communiceren wat hij precies wil. Daarnaast kan de business analist de informatie verkeerd interpreteren, of uitgaan vanuit de verkeerde veronderstellingen. Dit proces van miscommunicatie en incorrecte aannames kan zich bovendien herhalen tussen elke schakel, waardoor het probleem exponentieel groter wordt hoe verder in de keten van ontvangers gegaan wordt. Testers zijn vaak de laatste ontvangers van de informatie in de communicatiestroom, waardoor zij vaak de informatie ontvangen die het minst aansluit bij de verwachtingen van de klant.

Getrainde business analisten en projectmanagers kunnen dergelijke problemen voorkomen. Toch is het niet altijd vanzelfsprekend om op de talenten van de business analisten en projectmanagers te rekenen om te garanderen dat de *requirements* vanaf de eerste dag goed en duidelijk geformuleerd zijn, en dat het uiteindelijke ontwikkelde product daar dan ook naadloos op aansluit. [18]

Daarnaast is een ander probleem dat er een kloof zit tussen business en *development.* Business verwacht immers dat *developers* bepaalde *features* opleveren, maar hebben zelf weinig inzicht in de processen die aan het afgeleverde product voorafgaan, en al zeker niet aan bepaalde handelingen die er nog moeten op volgen. Denk bijvoorbeeld aan het verkleinen van technische schuld. Wanneer bepaalde deadlines bijvoorbeeld niet gehaald kunnen worden, is het ook moeilijk voor *developers* om precies aan te geven waar iets is misgegaan, aangezien het takenpakket van een *developer* zo technisch is. [19]

Een antwoord op bovengenoemde probleemstellingen is *Behavior-Driven Development*, waarop dieper ingegaan zal worden in dit hoofdstuk.

### 5.4.2 Roots in Test-Driven Development

BDD is ontstaan als een antwoord op de beperkingen van Test-Driven Development (TDD). TDD is immers erg technisch van aard, waardoor de businessdoelen vaak uit het oog verloren worden. Het schrijven van testen vergt immers heel wat technische kennis, en dat uit zich dan ook in de

uitwerking van deze testen. Terwijl deze testen leesbaar zijn voor *developers,* zijn deze testen vrijwel onleesbaar voor een leek. Dit maakt het moeilijk om aan de hand van testen te rapporteren naar business of klanten toe. Bovendien ligt de focus van één test doorgaans op één methode, en niet op een businessdoel, waardoor de kloof tussen *development* en business opnieuw vergroot. Een *developer* vraagt zich immers niet af of zijn test wel beantwoordt aan een specifieke eis van business, maar wel of de code doet wat hij moet doen. *Unit tests* worden doorgaans ook niet geschreven met het oog op de toekomst, maar steunen eerder op een zo eenvoudig mogelijk design om code te testen. Bovendien zijn *unit tests* ook gevoelig aan API-veranderingen. Wanneer de API verandert, moeten de overeenkomstige testen immers herschreven worden. [18]

*Behavior-Driven Development* probeert een oplossing te voorzien voor deze problemen.

### 5.4.3 Definitie van Behavior-Driven Development

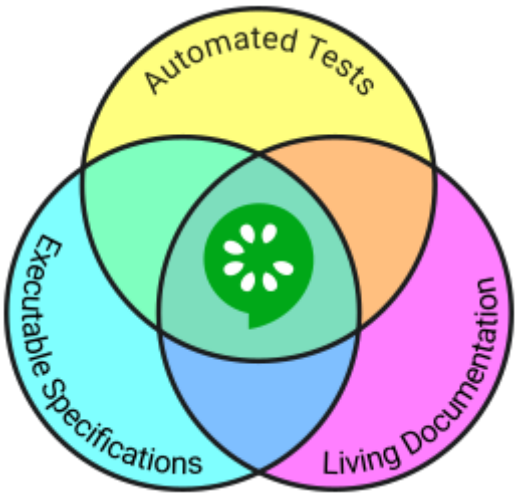
BDD is in 2003 ontstaan, toen een kleine groep mensen met een achtergrond *in extreme programming* (XP) op zoek ging naar manieren om TDD te verbeteren. Dan North, de grondlegger van BDD omschrijft het concept als volgt:

*"BDD is een second-generation, outside-in, pull-based, multi-stakeholder, multiple-scale, high-automation, agile methodologie.*

*Het beschrijft een cyclus van interacties met goed gedefinieerde outputs, resulterend in de levering van werkende, geteste software die ertoe doet."* [19]

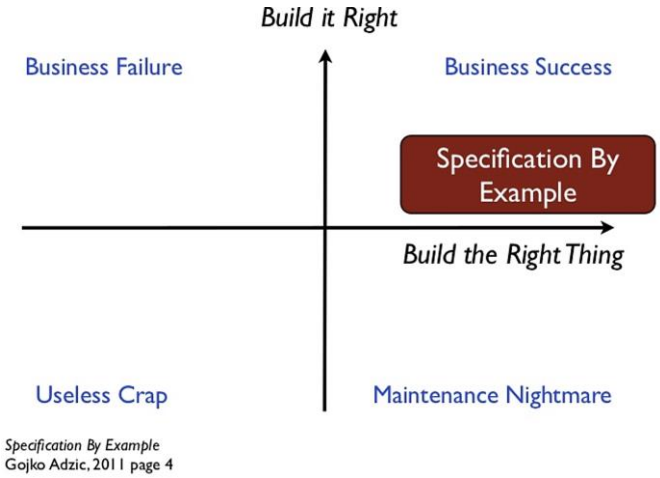
### 5.4.4 Cucumber

Dan North wou, toen hij BDD in het leven riep, een nieuw middel ontwikkelen om aan BDD te kunnen doen. Zijn idee was om (geautomatiseerde) *acceptance tests, functional requirements* en software documentatie in één nieuw formaat te combineren. Dit formaat moest bovendien begrijpelijk zijn voor mensen zonder een technische achtergrond. Tot slot moest dit nieuwe formaat ook gebruikt kunnen worden door *testing tools.* Dit idee heeft tot de creatie van Cucumber geleid. [20]




*figuur 66 - visuele weergave van de oorsprong van Cucumber*

Het heeft tot 2008 geduurd tot Cucumber eindelijk het daglicht zag. Het was uiteindelijk Aslak Hellesøy die bedacht had dat er een taal moest zijn, maar eveneens een bijhorend proces en een bijhorende tool om de visie van Dan North waar te maken. Deze tool moest gebruikt worden door technische en niet-technische mensen, en het idee was dat hier de "*single source of truth of software behaviour",* oftewel het enige punt van waarheid in het gedrag van de software, zou omschreven worden. Dit is hoe Cucumber ontstaan is. [20] Cucumber is dus een middel om BDD, het proces, te ondersteunen.




*figuur 67 - Specification by example foor Gojko Adžić*

In 2011 gaf Gojko Adžić, één van de specialisten in de materie, een beter passende naam aan BDD, namelijk "*Specification by Example."* [21]

*Specification by Example,* zoals beschreven door Adžić, bestaat voornamelijk uit twee activiteiten, namelijk *specification workshops,* meerbepaald de techniek van de *three amigos* en *outside-in development.* Beide activiteiten worden in volgende onderdelen omschreven.

### 5.4.5 Cucumber tests

Cucumber werkt aan de hand van scenario's. Een scenario wordt geschreven in Gherkin, en bestaat uit meerdere stappen die telkens een vaste structuur volgen. Een scenario volgt namelijk het *Given-When-Then-formaat*. Scenarios worden opgenomen in *feature files.* Een scenario in een *feature file* kan er dus als volgt uitzien:

```
Scenario: As a user, I want to be able to add VIP passengers to economy flights
   Given there is an economy flight
   When we have a VIP passenger
   Then you can add him but cannot remove him from an economy flight
```

Uiteraard kan dit scenario op zich geen testresultaten produceren. Het is de bedoeling dat er nu *glue code* geschreven wordt, die de eigenlijke code aan de *feature file* kan lijmen. Het mooie aan Cucumber is dat deze *glue code* in vrijwel elke taal geschreven kan worden. Dit betekent dat *developers* de *glue code* of testen gewoon kunnen schrijven in de taal waarin ze de rest van hun code

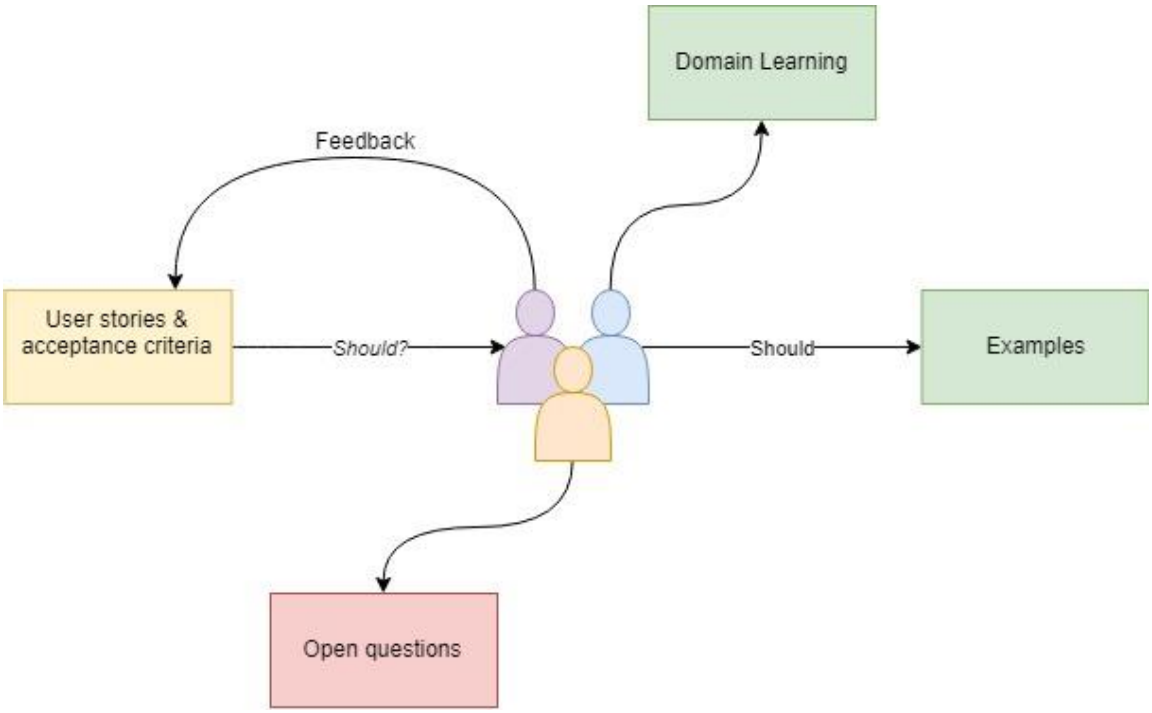
aan het schrijven zijn en dat vrijwel geen bijkomende kennis vereist is. *Glue code* in Java voor bovenstaand scenario kan dan bijvoorbeeld de volgende zijn:

```
@Given("^there is an economy flight$")
public void thereIsAnEconomyFlight() throws Throwable {
    economyFlight = new EconomyFlight("1");
}

@When("^we have a VIP passenger$")
public void weHaveAVIPPassenger() throws Throwable {
    john = new Passenger("John", true);
}

@Then("^you can add him but cannot remove him from an economy flight$")
public void youCanAddHimButCannotRemoveHimFromAnEconomyFlight() throws Throwable {
    assertAll("Verify all conditions for a VIP passenger and an economy flight",
            () -> assertEquals("1", economyFlight.getId()),
            () -> assertEquals(true, economyFlight.addPassenger(john)),
            () -> assertEquals(1, economyFlight.getPassengersList().size()),
            () -> assertEquals("John",
economyFlight.getPassengersList().get(0).getName()),
            () -> assertEquals(false, economyFlight.removePassenger(john)),
            () -> assertEquals(1, economyFlight.getPassengersList().size())
    );
}
```

### 5.4.6 The three amigos

Een *agile* techniek die eigen is aan BDD, en die kan helpen de kloof te dichten tussen business en *development* is het principe van *'The three amigos.'*

Deze 'drie vrienden' duiden drie specifieke rollen binnen het IT-team aan, namelijk een business analist of *product owner,* een *developer* en tot slot een tester. Deze drie figuren moeten vervolgens samen een beeld scheppen van wat de te ontwikkelen software uiteindelijk moet doen. Het is dus zinvol om deze techniek toe te passen aan het begin van een project, maar daarnaast kan deze werkwijze ook in de loop van het project toegepast worden, om te garanderen dat iedereen nog op dezelfde lijn zit, en dat de businessdoelen nog steeds correct worden nagestreefd. [19]




*figuur 68 - The three amigos*

De drie *amigos* gaan in eerste instantie samen aan de slag om de *user stories* te omschrijven, aan de hand van gebruiksvoorbeelden die ze kunnen bedenken in het kader van de te ontwikkelen software. Dit proces heet "*Example Mapping.*"

*User stories* worden traditioneel geschreven door de *product owner* of business analist. Toch kan dit proces, wanneer overgelaten aan één persoon met deze rol, een aantal problemen opleveren. Denk bijvoorbeeld aan volgende mogelijke struikelblokken [22]:

a) De *user stories* kunnen te beperkt zijn. Een mogelijke *user story* zou dan kunnen zijn "Ik wil geld op mijn rekening kunnen storten."
Aan de hand van deze omschrijving is het onmogelijk om te weten op welke manier geld gestort moet kunnen worden.
b) *User stories* kunnen te gedetailleerd zijn, en bijvoorbeeld erg specifieke randgevallen omschrijven.
c) *User stories* kunnen technisch onhaalbaar zijn. Dit is een probleem dat zich kan voordoen wanneer de product owner zelf geen of een beperkte technische achtergrond heeft.

Dergelijke problemen met de *user stories* bij de aanvang van een project kunnen heel snel een heleboel laten mislopen. De *three amigos* gaan dit probleem tegen, aangezien elke rol een unieke en gespecialiseerde kijk heeft op de *gehele software delivery pipeline*.

Hoewel het principe van *the three amigos* niet eigen is aan DevOps, past het wel perfect in dit kader, aangezien *collaboration, accountability* en *ownership* centraal staan. Bovendien wordt door deze werking enorm gestimuleerd dat elk teamlid op de hoogte is van de verantwoordelijkheden van andere teamleden, en het gebeuren tijdens het ontwikkelingsproces en *domain learning*.

#### 5.4.6.1 Example mapping

*Example mapping* is een *agile* techniek die haar oorsprong vindt in BDD. Het is een techniek die doorgaans wordt toegepast door de *tree amigos* als basis om de *acceptance criteria* vast te leggen. [23]

De *three amigos* gaan aan de slag met een set van vier gekleurde kaartjes, waarop geschreven kan worden.

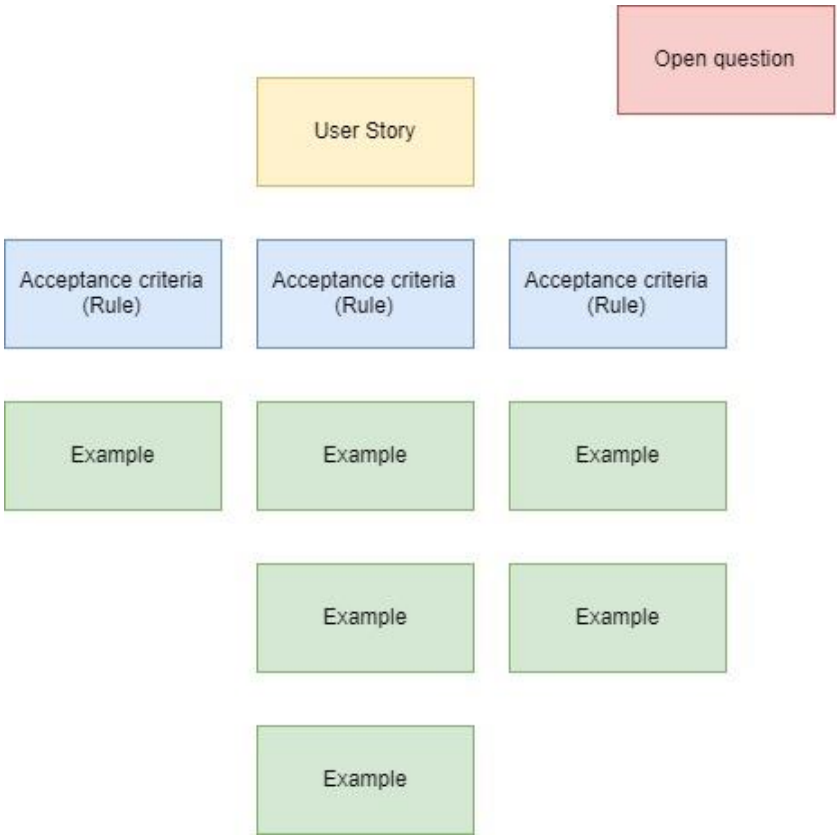
Allereerst wordt een *user story* op een geel kaartje geschreven, en dat kaartje wordt bovenaan een bord geplaatst.

Vervolgens kunnen de *rules,* oftewel *acceptance criteria,* die al bekend zijn op een blauw kaartje geschreven worden, en onder de corresponderende *user story* geplaatst worden.

Daarna gaan de *three amigos met elkaar* in dialoog om concrete voorbeelden te bedenken om de *rules* te illustreren. Deze concrete voorbeelden worden op een groen kaartje geschreven, en vervolgens onder de bijhorende *rule* op het bord geplaatst. Deze examples zijn in feite de *acceptance tests.*

Door het gesprek tussen de *three amigos* en het bedenken van concrete voorbeelden, kunnen na verloop van tijd enkele vragen naar boven komen, waarop niemand een duidelijk antwoord kan geven. Die vragen worden *open questions* genoemd, en worden vastgelegd op een rood kaartje, en dienen op een later moment verduidelijkt te worden, door bijvoorbeeld de klant.

Dit proces blijft zich herhalen tot iedereen tevreden is en de *scope* van het project duidelijk en afgebakend is, of tot er voldoende tijd verstreken is. Zo is het immers aangeraden om niet langer dan vijfentwintig minuten te spenderen aan een *story.* Is dit in de praktijk toch het geval, dan is het team waarschijnlijk nog niet klaar om deze manier van werken op te nemen. [23]




*figuur 69 - Example mapping*

Door aan *example mapping* te doen, kan het team onmiddellijk een beeld scheppen van hoe goed ze de *requirements* hebben kunnen vastleggen. Hangt het bord vol rode kaartjes, oftewel *open questions,* dan is het duidelijk dat er nog veel te leren valt over de betrokken *user story.* Zijn er veel blauwe kaartjes, oftewel *rules* of *acceptance criteria,* dan is de *user story* waarschijnlijk groot en ingewikkeld. In dat geval is het aangeraden om deze *user story* in kleinere *stories* te verdelen. Zijn er veel groene kaartjes of voorbeelden gekoppeld aan een *story,* dan is de betrokken *user story* mogelijk enorm complex. In dat geval moeten de *rules* mogelijk in meerdere stukken onderverdeeld worden. [23]

Het doel van *example mapping* is om de *core behavior* van het te ontwikkelen product vast te leggen. Daarbij is het ook belangrijk om aanvankelijk gericht te werk te gaan, en de aandacht uitsluitend op de essentie te vestigen. *Example mapping* zorgt er dus voor dat te grote *stories* niet in een sprint terechtkomen. [23]

Net zoals omschreven bij *the three amigos,* past *example mapping* eveneens in het kader van DevOps, aangezien *collaboration, domain learning, accountability* en *ownership* gestimuleerd worden.

Tot slot is het belangrijk om niet in een aantal valkuilen te vallen bij *example mapping.* Zo is het zeker niet de bedoeling om de acceptatietesten, namelijk de *examples* op de groene kaartjes, ook al meteen in Gherkin te schrijven. Het doel van *example mapping* is om een duidelijk beeld te scheppen van de te ontwikkelen software, en niet om testen te genereren. Dat is wel mogelijk voor een gevorderd team, maar is zeker geen vereiste.

### 5.4.7 Outside-In Development met Cucumber

Zoals de naam aangeeft, is *outside-in development* een verantwoordelijkheid van de *developers.* Eens de Cucumber scenario's geschreven zijn, is het aan de ontwikkelaars om hiermee aan de slag te gaan. Dit betekent dat zij aan de hand van een Cucumber-scenario gaan bepalen wat er geïmplementeerd moet worden. Vervolgens schrijven zij de code en koppelen deze met de nodige testen oftewel *glue code* aan het Cucumber scenario. [19] Dit proces herhaalt zich, tot de *feature* werkt zoals beschreven was.

Deze methode heet "*outside-in development*" omdat de ontwikkelaars hier doorgaans beginnen te werken aan de functionaliteit die het dichtst bij de gebruiker staat, zoals de gebruikersinterface. Met andere woorden, beginnen ze met de functionaliteit aan de buitenkant van het systeem, en werken vervolgens naar de kern toe. [20]

Net zoals bij TDD, worden de ontwikkelaars gestuurd door (aanvankelijk) falende testen, om de code te schrijven. Het verschil met BDD en met Cucumber, is dat er gewerkt wordt op een zeer hoog en abstract niveau, en dat er geen focus ligt op specifieke methodes of *classes.* [18]

## 5.5 Test Management met Xray

### 5.5.1 Inleiding

Zoals omschreven in de uitwerking van de stageopdracht, kan een *software delivery team* dankzij Xray rechtstreeks in Jira aan *test management* doen. Xray maakt dit mogelijk door nieuwe *issue types* te voorzien. Bovendien kunnen rapporten ook rechtstreeks in Jira gegenereerd worden, waardoor het enorm makkelijk worden om projecten te managen, voortgang en problemen op te volgen en efficiënt naar klanten te rapporteren.

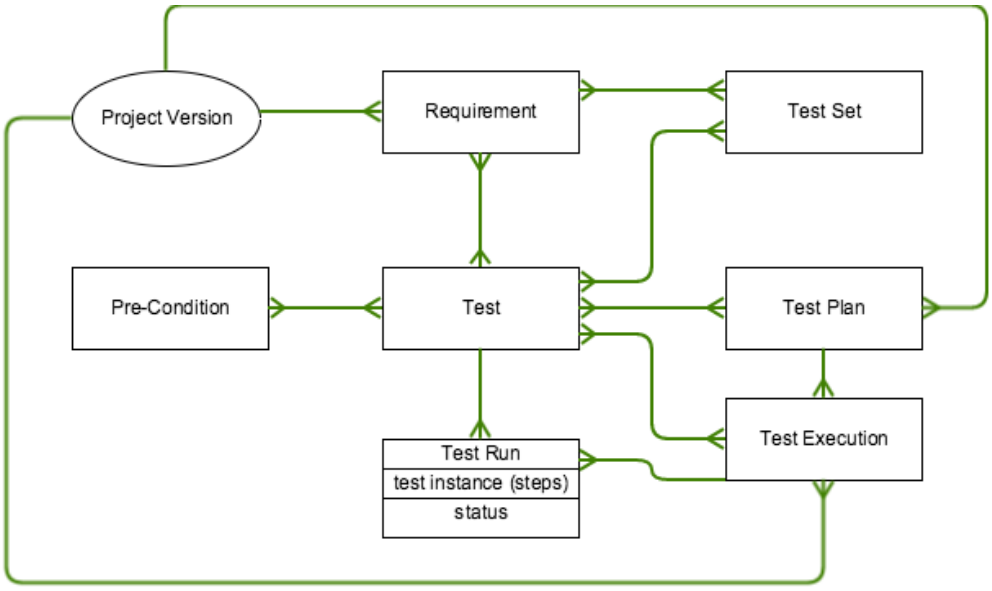
### 5.5.2 Werking custom issue types

Xray voorziet het *test issue type*, waarin rechtstreeks testen geschreven kunnen worden. Dit kunnen manuele testen zijn, automatische testen (die bijvoorbeeld geschreven zijn in Selenium), of Cucumber-testen.

Een test kan vervolgens in een *test set* gestoken worden. Een *test set* is een manier om testen te groeperen en te ordenen, om het overzicht beter te kunnen bewaken.

Testen kunnen daarnaast ook in een *test plan* gestoken worden. Een *test plan* is gekoppeld aan een bepaalde *fix version.* Op deze manier kunnen meerdere testen voor bijvoorbeeld een bepaalde sprint efficiënt ingepland worden.

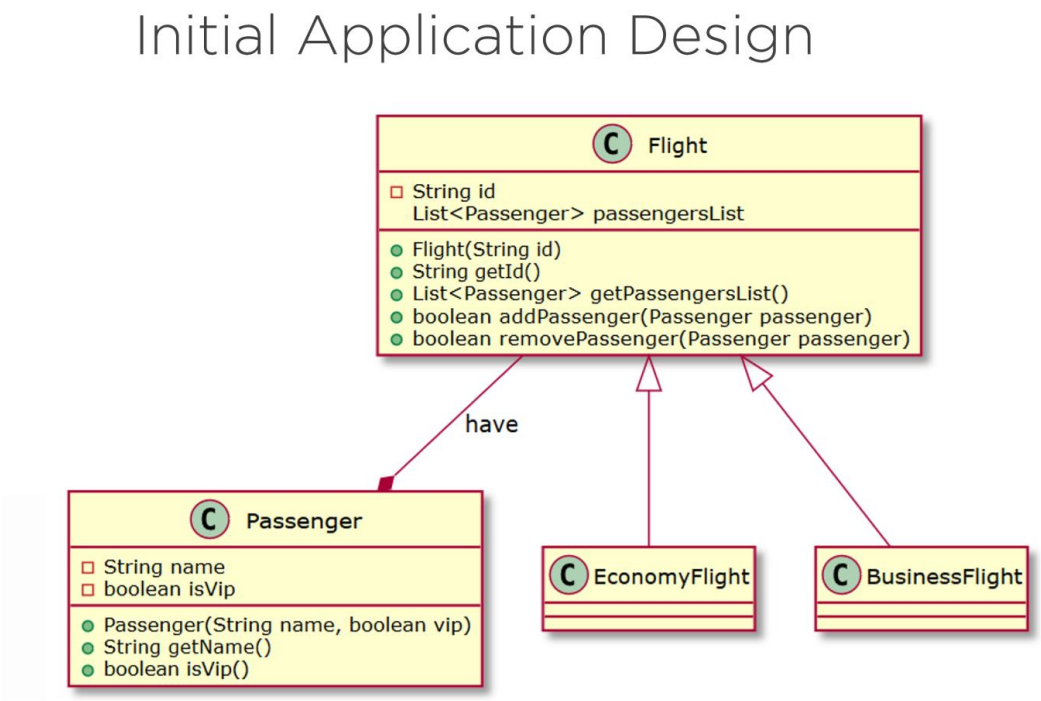
Een *test plan* wordt vervolgens best in een *test execution* gestoken. Op deze manier kunnen testen effectief uitgevoerd worden. Zoals verder in dit document beschreven is, kan een *test execution* dan opgenomen worden in de *CI-pipeline,* en kunnen de testen automatisch in de *repository* geplaatst worden. Eveneens wanneer bijkomende testen in de loop van het project geschreven worden kunnen testresultaten bij elke *build* geüpload worden naar Jira.




*figuur 70 - Entity Relationship Diagram (ERD) van Xray issues*

### 5.5.3 Opbouw Xray project

Om een concrete implementatie van Xray en BDD met Cucumber-testen te illustreren, zal er gewerkt worden met een concrete *repository* met Java code, zoals voorgesteld op het *Unified Modeling Language diagram (UML)* hieronder.




*figuur 71 – UML-diagram airport project*

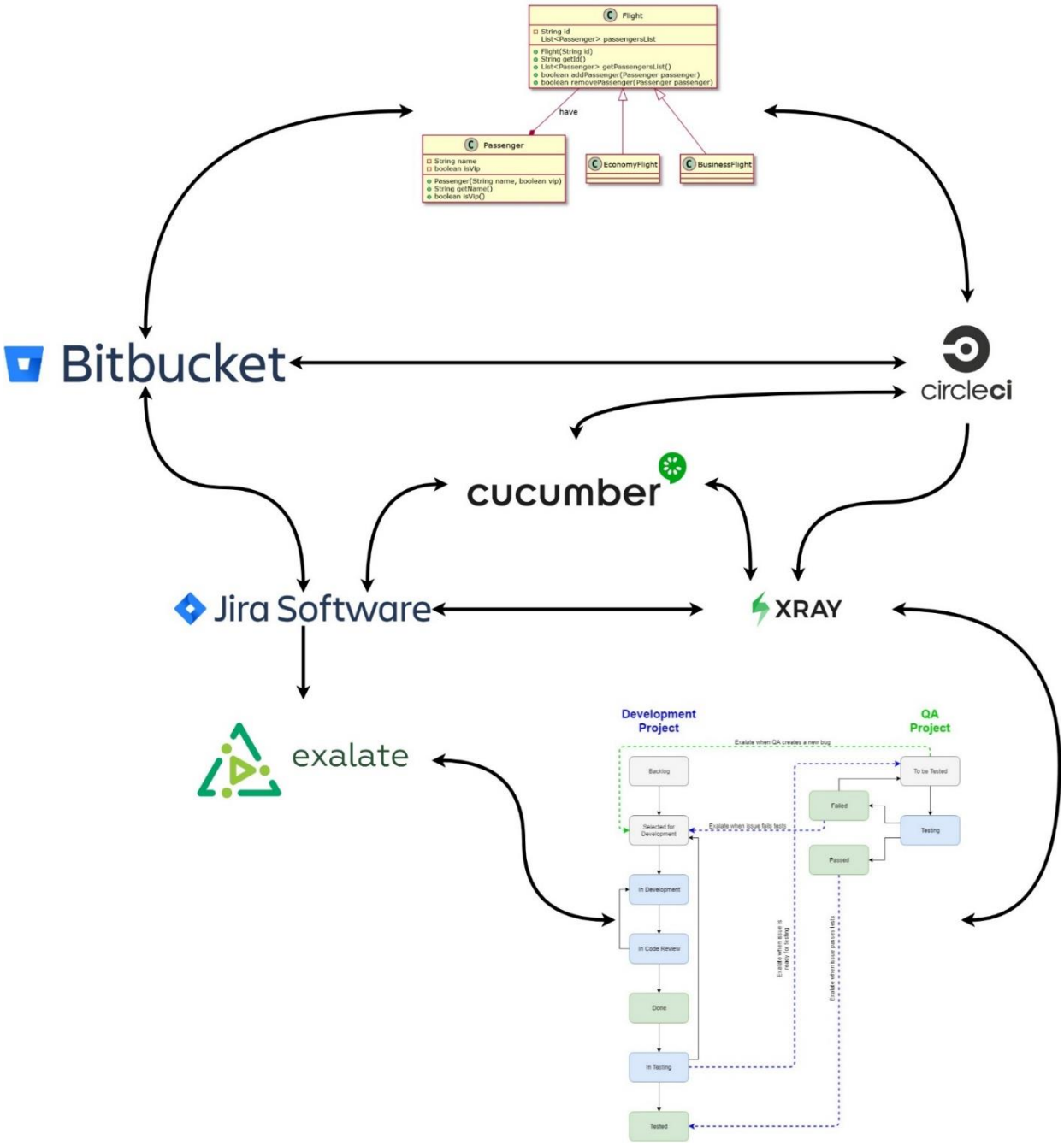
Verder zijn er ook twee projecten opgesteld in Jira. Een project voor *development* en een project voor QA, waarin de *test issues* zullen worden aangemaakt. Deze scheiding tussen beide projecten gebeurt om het overzicht voor het *development team* te bewaren, door te voorkomen dat het kanbanbord overbeladen wordt met *test issues, test sets, test plans en test executions*, terwijl deze niet moeten opgevolgd worden door *developers*.

*Epics* en *stories* dienen initieel aangemaakt te worden in het *development project,* en vervolgens via Exalate gesynchroniseerd te worden met het QA-project. Hier kunnen vervolgens de nodige *test issues* aangemaakt worden, en kan het schrijven van de testen beginnen. De werking van Exalate en het synchroniseren van *issues* tussen een *development project* en een QA-project is volledig uitgeschreven binnen de uitwerking van de stageopdracht.

De *repository* is vervolgens via Bitbucket aan beide projecten gekoppeld.

Bitbucket is dan weer gekoppeld aan CircleCI, om continuous integration waar te maken, en de testen bij elke *build* te runnen, en vervolgens de testresultaten naar Jira te posten.

In dit voorbeeld worden Cucumber-testen rechtstreeks in Jira geschreven en vervolgens via CircleCi naar de *repository* gestuurd. Wanneer de testen dan vervolledigd worden in de *repository*, worden de testresultaten automatisch naar Jira doorgestuurd.




*figuur 72 - Visuele voorstelling van de implementatie van Xray in de CI-pipeline*

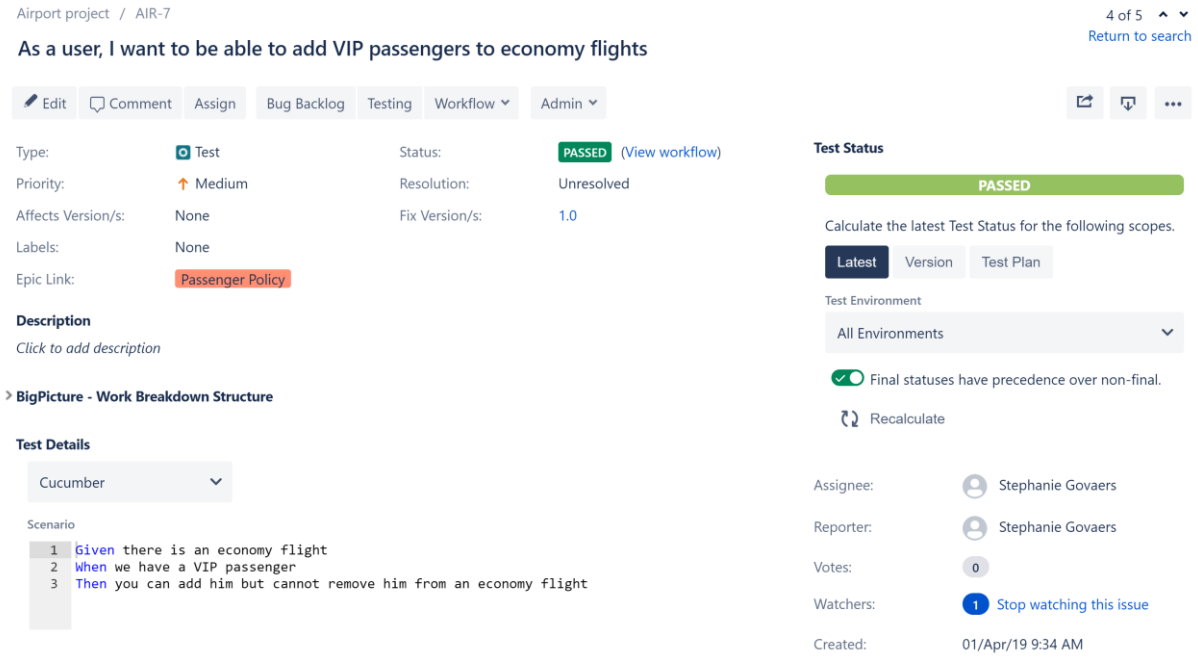
Een belangrijke opmerking voor een Java-project, is dat we Cucumber moeten kunnen aanspreken. Dit kan bijvoorbeeld door een Maven-project te gebruiken en de juiste *dependencies* op te nemen in de *pom-file*. Deze dependencies zijn dan de volgende:

```
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>2.3.1</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.cucumber</groupId>
```

```
    <artifactId>cucumber-junit</artifactId>
    <version>2.3.1</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>pro-plugin</artifactId>
    <version>2.1.0</version>
    <scope>test</scope>
</dependency>
```

### 5.5.4 Schrijven van Cucumber-testen in Jira

Het sterkste punt van Xray, waarmee een duidelijk onderscheid van de concurrentie wordt gemaakt, is dat naast manuele testen en andere geautomatiseerde testen, Cucumber-testen rechtstreeks in Jira geschreven kunnen worden. De BDD-processen en technieken die hieraan voorafgaan, worden omschreven in het hoofdstuk van BDD. In dit onderdeel zal nu omschreven worden hoe deze processen zich concreet kunnen uiten in Cucumber-testen.




*figuur 73 - Test issue met Cucumber scenario*

Op bovenstaande afbeelding is een *test issue* van Xray zichtbaar. Zoals afgebeeld, is hier rechtstreeks in het *issue ticket* een Cucumber scenario geschreven. Door de koppeling met Bitbucket en de integratie in CircleCI kunnen deze testen bij elke *pull* opgehaald worden, vertaald worden naar een *cucumber feature file*, en vervolgens in de *repository* geplaatst worden. Een dergelijke gegenereerde *feature file* ziet er dan uit zoals de afbeelding hieronder.

```
@AIR-14
Feature: Test Execution for 1.0

    @TEST_AIR-44 @TESTSET_AIR-12
    Scenario: I want to be able to register on the webpage of the airport
        Given there is a register button on the homepage
        When I click the register button
        Then I am redirected to the registration form
    @TEST_AIR-11 @TESTSET_AIR-12
    Scenario: As a user, I want to be able to add VIP passengers to business flights
        Given there is an business flight
        When we have a VIP passenger
        Then you can add him but cannot remove him from a business flight
    @TEST_AIR-10 @TESTSET_AIR-12
    Scenario: As a user, I want to be able to add/remove usual passengers to/from business flights
        Given there is an business flight
        When we have a usual passenger
        Then you cannot add or remove him from a business flight
    @TEST_AIR-7 @TESTSET_AIR-12
    Scenario: As a user, I want to be able to add VIP passengers to economy flights
        Given there is an economy flight
        When we have a VIP passenger
        Then you can add him but cannot remove him from an economy flight
    @TEST_AIR-6 @TESTSET_AIR-12
    Scenario: As a user, I want to be able to add/remove usual passengers to/from economy flights
        Given there is an economy flight
        When we have a usual passenger
        Then you can add and remove him from an economy flight
```

*figuur 74 - Cucumber feature file gegenereerd door een Test Execution*

Om een test op te nemen in een *CI-tool* zoals CircleCI, moet er gewerkt worden met *curls* die de API van Xray aanspreken. In een dergelijke curl moet het Jira-ID van het overeenkomstige *issue* opgenomen worden. Het is dus noodzakelijk dat alle testen opgenomen worden in een *test execution issue type* van Xray, zodat er maar één ID nodig is om een groot aantal testen aan te spreken. Voor *CI-tools* zoals Jenkins en Gitlab is de integratie van Xray veel eenvoudiger in te stellen, maar het is hoe dan ook om testen toch te groeperen in *test sets, test plans* en *test executions*.

Voor CircleCI worden alle curls mee opgenomen in de *config.yml* file in de repository. Een curl die, zoals in bovenstaand voorbeeld de *test execution* met ID 'AIR-14' wilt aanspreken ziet er dan als volgt uit:

```
- run: 'curl -H "Content-Type: application/json" --output featureBundle.zip -X GET
-H "Authorization: Bearer ${token}" "https://xray.cloud.xpand-
it.com/api/v1/export/cucumber?keys=AIR-14"'
```

Alle (nieuwe) Cucumber-testen zullen dan als een gecomprimeerde map gedownload worden op een *virtual machine* (VM) van CircleCI. Er is vervolgens een *machine user* van Github nodig om de nodige git commands te kunnen uitvoeren en de map te downloaden en uitpakken in de repository.

Eens de *feature file* in de *repository* staat, kunnen de *developers* de *glue code* schrijven. Dergelijke code voor het gegeven voorbeeld ziet er dan als volgt uit:

```java
@Given("^there is an economy flight$")
public void thereIsAnEconomyFlight() throws Throwable {
    economyFlight = new EconomyFlight( id: "1");
}

@When("^we have a VIP passenger$")
public void weHaveAVIPPassenger() throws Throwable {
    john = new Passenger( name: "John",  vip: true);
}

@Then("^you can add him but cannot remove him from an economy flight$")
public void youCanAddHimButCannotRemoveHimFromAnEconomyFlight() throws Throwable {
    assertAll( heading: "Verify all conditions for a VIP passenger and an economy flight",
            () -> assertEquals( expected: "1", economyFlight.getId()),
            () -> assertEquals( expected: true, economyFlight.addPassenger(john)),
            () -> assertEquals( expected: 1, economyFlight.getPassengersList().size()),
            () -> assertEquals( expected: "John", economyFlight.getPassengersList().get(0).getName()),
            () -> assertEquals( expected: false, economyFlight.removePassenger(john)),
            () -> assertEquals( expected: 1, economyFlight.getPassengersList().size())
    );
}
```

Aangezien we werken met Java, hebben we een *testrunner class* nodig om de testen te kunnen uitvoeren. Bovendien moet er in deze klasse ook aangegeven worden dat de testresultaten als json moeten geformatteerd worden, aangezien dit een vereiste is om de testresultaten via Xray te kunnen uploaden.

*figuur 75 - Glue code geschreven in Java*

```java
import cucumber.api.CucumberOptions;
import cucumber.api.SnippetType;
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;

/**
 * Entry point for running the Cucumber tests in JUnit.
 */
@RunWith(Cucumber.class)
@CucumberOptions(
    plugin = {"pretty", "json:target/cucumber.json"},
    snippets = SnippetType.CAMELCASE,
    features = "classpath:features")
public class CucumberTest {

    /**
     * This class should be empty, step definitions should be in separate classes.
     */

}
```

*figuur 76 - Testrunner class*

*De config file van CircleCI ziet er dan (gedeeltelijk) als volgt uit:*

```
- run: mvn clean install -B

- run: mvn test # run the actual tests

- save_cache: # saves the project dependencies
    paths:
```

```
      - ~/.m2
    key: Airport-Demo

- store_test_results: # uploads the test metadata from the `target/surefire-
reports` directory so that it can show up in the CircleCI dashboard.
    path: target/surefire-reports

- run:
    name: Get API token
    command: |
      echo export token=$(curl -H "Content-Type: application/json" -X POST --data
"{ \"client_id\": \"$client_id\",\"client_secret\": \"$client_secret\" }"
https://xray.cloud.xpand-it.com/api/v1/authenticate| tr -d '"') >> $BASH_ENV
      source $BASH_ENV

- run: 'curl -H "Content-Type: application/json" --output featureBundle.zip -X GET
-H "Authorization: Bearer ${token}"  "https://xray.cloud.xpand-
it.com/api/v1/export/cucumber?keys=AIR-14"'

- run: "unzip -o featureBundle.zip -d /home/circleci/Airport-
Repository/src/test/resources/features"

- run: "git config --global user.email $git_email"

- run: "git config --global user.name $git_user"

- run: cd /home/circleci/Airport-Repository/ && sh scripts/push_tests_github.sh

- run: 'curl -H "Content-Type: application/json" -H "Authorization: Bearer $token"
--data @target/cucumber.json  "https://xray.cloud.xpand-
it.com/api/v1/import/execution/cucumber?keys=AIR-14"'
```

*figuur 77 – config.yml om testen te downloaden en resultaten te uploaden naar Jira via CircleCI en Xray*

## 5.6 Overzicht toegepaste DevOps-principes

| Omschrijving principe | | Verklaring |
|---|---|---|
| Accountability, ownership & end-to-end responsibility | ✔ | Door het toepassen van *Behavior-Driven Development* wordt de focus gelegd op de businessdoelen. Bovendien wordt er gebruik gemaakt van een eenvoudige taal die voor iedereen verstaanbaar is. Hierdoor vergroot de kennis van elk teamlid van de realisaties die dienen te gebeuren, terwijl minder technische profielen voordien minder betrokken werden in de *end-to-end pipeline*. Hierdoor is er ook een mogelijkheid om andere profielen verantwoordelijkheden te geven doorheen de hele *pipeline*, terwijl dat voordien misschien maar voor een onderdeel van de *pipeline* mogelijk was. |
| Collaboration | ✔ | *Behavior-Driven Development* bevordert de samenwerking enorm. Denk bijvoorbeeld aan *the three amigos* en *example mapping* om de requirements vast te leggen en de *backlog* samen te stellen. Alle rollen binnen een team werken intensief samen om dat te kunnen realiseren. Business analisten of *product owners* kunnen door de eenvoudige taal binnen BBD zelf testen schrijven, die vervolgens door de *developers* met code aangevuld worden. |
| Automation | ✔ | Door gebruik te maken van een repository op Bitbucket, enerzijds gekoppeld aan Jira, met een integratie van Xray, en anderzijds met CircleCI als *CI-tool*, kunnen Cucumber-testen rechtstreeks in Jira geschreven worden, om vervolgens automatisch en rechtstreeks in de testklassen van de ontwikkelaars toegevoegd te worden. Wanneer zij de testen koppelen aan hun code via de nodige *glue code,* worden de testresultaten bij elke *pull request* en *merge* dankzij de API van Xray en CI geüpload naar Jira, waar de testresultaten zichtbaar worden, en waar vervolgens testrapporten gegenereerd kunnen worden. Tot slot is er met Xray ook de mogelijkheid om een gelijkaardig scenario te realiseren met andere automatische testen, zoals Selenium of Protractor. Hierover zijn echter geen concrete implementaties gedaan binnen deze stage of deze paper. |
| Continuous Integration | ✔ | Zoals hierboven vermeld, is Continuous Integration gerealiseerd aan de hand van Xray, Bitbucket en CircleCI. Deze tools werken samen om de Cucumber-testen automatisch uit te voeren telkens er een *feature-branch* wordt aangemaakt en bij elke *pull request* of *merge*. |
| Continuous Improvement | ✔ | Bovenstaande realisaties kunnen allemaal in het teken staan van Continuous Improvement. Door deze implementatie neemt de taaklast van het team immers af door automatisatie. Daarnaast is BDD ook een verbetering ten opzichte van Test-Driven Development door bijvoorbeeld de nadruk te leggen op de businessdoelen, door een vereenvoudigde taal te gebruiken, enzovoort. Naast deze interne verbeteringen, kunnen deze aanpassingen ook naar externen een enorme verbetering zijn. Door de vereenvoudigde taal is het eenvoudiger om de stand van zaken naar klanten toe te communiceren, maar ook om de noden van de klant vast te leggen. |
| Fast & reliable deploys with Continuous Delivery | ✗ | Zoals eerder omschreven valt CD buiten de scope van dit project. Het is echter zo dat bovenstaande DevOps-realisaties een ideale basis zijn om CD op een optimale manier te kunnen implementeren. |

# Conclusie

In het kader van het ontstaan van Appmind binnen Bewire, moest een *software delivery pipeline,* bestaande uit de nodige *tools* en ondersteunde processen, onderzocht en gedefinieerd worden. Het doel was om met deze *pipeline* aan een hoge snelheid mobiele applicaties te ontwikkelen, zonder te moeten inboeten aan kwaliteit. Bovendien moesten voor een aantal concrete probleemstellingen de gepaste *tools* geadviseerd worden.

Allereerst is er onderzoek verricht naar het optimaliseren van de activiteiten binnen de Atlassian stack. Er is automatisatie opgezet binnen de workflows van Jira. Daarnaast moest er een middel zijn om, aan de hand van de backlog in Jira, roadmaps samen te stellen. Na analyse en experimentatie met verschillende tools, blijkt Portfolio for Jira de tool te zijn die het meeste aansluit bij de noden van Bewire. Verder moest ook nagegaan worden hoe permissies binnen Jira correct moesten ingesteld worden, zodat werknemers enkel toegang hebben tot de projecten waarbij ze zelf betrokken zijn. Hiervoor is een handleiding geschreven die in bijlage is opgenomen. Verder is ook omschreven hoe er moet omgegaan worden met GDPR binnen de Altassian stack. Aan de hand van Exalate, een plug-in voor Jira, kunnen verschillende Jira-projecten en zelfs Jira-instanties met elkaar gesynchroniseerd worden. Dit betekent dat, indien het wenselijk is, Bewire-consultants externe projecten kunnen synchroniseren met hun eigen bestaande, geoptimaliseerde en geautomatiseerde workflows. Consultants kunnen op deze manier tijd en dus geld besparen door geen externe, complexe workflows te moeten aanleren, en moeten enkel gebruik maken van de *issues* die voor hun van belang zijn. Bovendien kan Exalate ook nuttig zijn wanneer Bewire zou werken met onderaannemers. Bewire heeft bijvoorbeeld zelf geen QA-team, terwijl dat voor bepaalde projecten mogelijk toch wenselijk kan zijn. Dankzij Exalate kan Bewire toegang verlenen tot bepaalde *issues* voor externen, en kunnen workflows van alle betrokken partijen zo gestroomlijnd worden.

In het kader van productkwaliteit is Behavior-Driven Development grondig onderzocht, zijn zinvolle methodieken omschreven en is er een implementatie gebeurd met Xray om binnen Jira aan testmanagement te doen. Hierbij is een praktisch opstelling gebeurd met geautomatiseerde Cucumber-testen binnen een *CI-pipeline* met Bitbucket en CircleCI, waarbij ook automatische testrapporten gegenereerd worden.
Verder zijn ook een aantal *code quality tools* onderzocht, waarbij zowel SonarQube als Code Climate als potentiële tools aangeraden zijn, afhankelijk van de noden van het project in kwestie.

Tot slot werd ook Jira Service Desk onderzocht, geïmplementeerd en geautomatiseerd in het kader van support.

Met het oog op bovengenoemde probleemstelling, werd volgende onderzoeksvraag geformuleerd: “Hoe kunnen DevOps-principes ingezet worden om de kwaliteit van projectmanagement en de projectkwaliteit te verbeteren in kortlopende projecten voor mobile development?” Vervolgens werden de volgende DevOps-principes toegepast op de gerealiseerde implementaties: *accountability, ownership, end-to-end responsibility, collaboration, automation, continuous integration* en tot slot *continuous improvement.*

# Bibliografie


[1] L. Filippone, *Stageopdracht: Stage Mobile Startup - Software Management Strategy,* 2018.

[2] R. Visser, „Roadmap: een handzame en doordachte projectplanning," Twynstra Gudde, [Online]. Available: https://www.twynstragudde.nl/roadmap-een-handzame-en-doordachte-projectplanning. [Geopend 20 maart 2019].

[3] B. Davies, *Making Portfolio for JIRA Work for Your Team - Atlassian Summit Europe 2017,* Atlassian, 2017.

[4] ConXioN, „Wat is GDPR - General Data Protection Regulation," ConXioN, 2018. [Online]. Available: https://gdpr-eu.be/wat-is-gdpr/. [Geopend 19 maart 2019].

[5] Atlassian, „Atlassian's GDPR Commitment," Atlassian, 2019. [Online]. Available: https://www.atlassian.com/trust/privacy/gdpr. [Geopend 21 maart 2019].

[6] R. Bellairs, „What is Code Quality? And How to Improve It," Perforce, 24 januari 2019. [Online]. Available: https://www.perforce.com/blog/qac/what-code-quality-and-how-improve-it. [Geopend 27 maart 2019].

[7] C. Jones, „Software Defect Origins and Removal Methods," Namcook Analytics LLC, 2012.

[8] G. Warren, „Code metrics values," Microsoft, 11 februari 2018. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2019. [Geopend 8 april 2019].

[9] B. Helmkamp, *Code Quality Lessons Learned,* Code Climate, 2016.

[10] H.-J. Buist, „Wat is technische schuld?," Computerworld, 5 januari 2017. [Online]. Available: https://computerworld.nl/development/96036-wat-is-technische-schuld. [Geopend 8 april 2019].

[11] D. Turkyilmaz, „Wat is DevOps?," Eduvision, 2015. [Online]. Available: https://www.eduvision.nl/blog/devops. [Geopend 20 mei 2019].

[12] J. Humble, „Continous Delivery," InformIT, 2010. [Online]. Available: https://continuousdelivery.com/. [Geopend 20 mei 2019].

[13] S. Pittet, „Learn Continuous Integration with Bitbucket Pipelines," Atlassian, [Online]. Available: https://www.atlassian.com/continuous-delivery/tutorials/continuous-integration-tutorial. [Geopend 4 3 2019].

[14] M. Fowler, „Continuous Integration," 2016. [Online]. Available: https://www.martinfowler.com/articles/continuousIntegration.html. [Geopend 4 March 2019].

[15] XpandIT, Quick Start Guide for Test Management, Lisabon: XpandIT, 2016.

[16] S. McConnell, Code Complete, 2004.

[17] Xpandit, Delivering better software using Test Automation, Londen, 2019.

[18] C. Tudose, *Java: BDD Fundamentals,* Pluralsight, 2019.

[19] S. Rose, *An Introduction to Behavior-Driven Development (BDD) With Cucumber for Java,* Sydney: Oracle Developers, 2015.

[20] A. Hellesøy, *The world's most misunderstood collaboration tool,* Cucumber, 2014.

[21] G. Adžić, Specification by Example: How Succesful Teams Deliver the Right Software, Manning, 2011.

[22] J. Farrel, *The Mobile Surge: How Cucumber Helps Your Three Amigos Speak Pidgin,* Disqus.

[23] M. Wynne, „Introducing Example Mapping," Cucumber, 2016. [Online]. Available: https://cucumber.io/blog/example-mapping-introduction/. [Geopend 26 maart 2019].

[24] G. Kim, J. Humble, P. Debois en J. Willis, The DevOps Handbook - How to create world-class agility, reliability & security in technology organizations, Portland: IT Revolution Press, 2016.

[25] L. Banica, M. Radulescu, D. Rosca en A. Hagiu, „Is DevOps another Project Management Methodology?," *Informatica Ecomomica,* vol. 21, nr. 3/2017, p. 39, 2017.

[26] B. Rockwood, *The DevOps Transformation: From here to there and why,* Joyent, Inc, 2011.

# III. Reflectie

Na twaalf fantastische weken kan ik deze stageperiode en het bijhorende onderzoek afronden. Ik kan oprecht zeggen dat ik tevreden ben met het resultaat en dat ik meer dan ooit dankbaar ben de kans gekregen te hebben om deze mooie uitdaging aan te gaan. Ik heb mijn kennis en vaardigheden kunnen aanscherpen en ik ben ongetwijfeld ontzettend gegroeid op professioneel vlak. Hoewel ik mezelf enorm heb ingezet om de nodige resultaten te kunnen bereiken, zou het allemaal niet mogelijk geweest zijn door de intensieve begeleiding van mijn bedrijfspromotor, Luca. Als ware mentor heeft hij ontzettend veel kennis en belangrijke inzichten gedeeld.

Deze stageperiode bij Bewire was ook enorm verrijkend door het fantastische team dat achter Bewire staat. Vanaf de eerste dag werden wij, als stagiairs, opgenomen in het team en als collega's beschouwd. De open cultuur en de mogelijkheid tot kruisbestuiving is voor mij ontzettend belangrijk. Wanneer ik niet meteen een oplossing vond voor bepaalde problemen, kreeg ik onmiddellijk hulp van één van de experts binnen Bewire. Bovendien wordt er heel wat kennis gedeeld binnen Bewire. Hiervoor wordt bijvoorbeeld elk kwartaal een B-Inspired-dag georganiseerd. Tijdens deze dag delen enkele Bewire-medewerkers hun inzichten op het podium, naar het concept van TED Talks. Hoewel deze evenementen net om die reden georganiseerd worden, is het binnen de muren van het kantoor duidelijk dat deze mindset ook tijdens de werkuren gehanteerd wordt, en dat er dagelijks kennis gedeeld wordt en dat collega's tijd vrijmaken om elkaar te helpen en dingen bij te leren. Ik kan enkel bewondering hebben voor een bedrijf waarin de plaats en ruimte wordt gegeven om van elkaar te leren, en waar er ook écht geluisterd wordt naar wat de medewerkers te zeggen hebben. Ik vind dit zelf ontzettend belangrijk, en zou zelf niet in een andere werkomgeving terecht willen komen.

Verder heb ik het gevoel dat ik ook als persoon ben kunnen groeien. Eén van de dingen die ik ontdekt heb is dat ik net zo graag zelfstandig werk als in een team, en die mogelijkheid heb ik ook gekregen tijdens mijn stage. Uiteraard heb ik deze stageopdracht niet alleen uitgevoerd, maar heb ik mogen samenwerken met mijn collega, Mathias. Langs de ene kant hebben we dus heel wat werk moeten verdelen en op elkaar moeten rekenen. In het verleden heb ik echter al heel wat projecten samen met Mathias afgerond, dus we hebben al leren samenwerken. Bovendien wisten we ook van elkaar waarin elk van ons goed is, en waarmee we mogelijk problemen hebben, en we hebben dan ook geprobeerd om hierop in te spelen. Hieruit neem ik mee dat het ontzettend belangrijk is om bewust na te denken over de sterktes van het team waarvan ik deel zal uitmaken. Verder heb ik ook geleerd om zelfstandig beslissingen te nemen, ondernemingszin te tonen en verantwoordelijkheid te nemen. De stageopdracht was immers vrij breed en moest nog concreet ingevuld worden. Hiervoor heb ik enorm veel vrijheid gekregen. Ik kreeg de kans om zelf met voorstellen naar voor te komen, software en methodieken uit te testen, enzovoort. Daarnaast heb ik ook de kans gekregen om ook technische ervaring op te doen. Deze uitdaging vond ik enorm motiverend. Bovendien was ik enorm verrast en dankbaar dat ik ook deze kans gekregen heb, en dat er ook echt naar mijn ideeën en adviezen geluisterd is. Hierdoor ben ik nog meer kunnen groeien, zowel op vlak van kennis en vaardigheden, maar ook als persoon. Ik heb immers geleerd om uitdagingen aan te gaan, over de grenzen van mijn comfortzone te gaan, om vertrouwen te hebben in mijn eigen kunnen, en om verantwoordelijkheid op te nemen voor eventuele fouten die ik maak.

# Bijlagen

## A. Handleiding User Groups & Project Roles in Jira

### A. Handleiding User Groups & Project Roles in Jira

# Permission schemes: User groups & Project roles

## 1 Probleemstelling:

Hoe kunnen we ervoor zorgen dat een bepaald type user enkel de projecten kan zien waarin hij betrokken is, terwijl andere projecten onzichtbaar zijn?

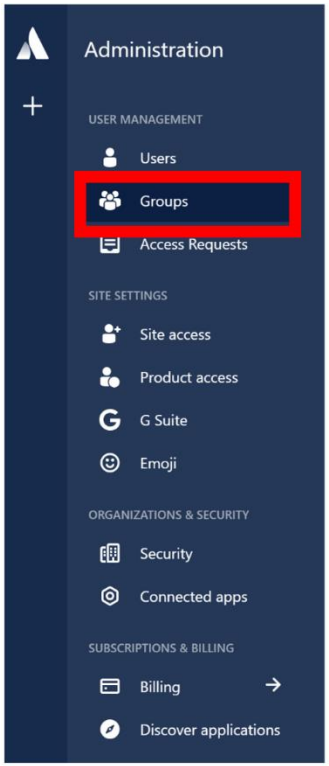
## 2 Samengevat

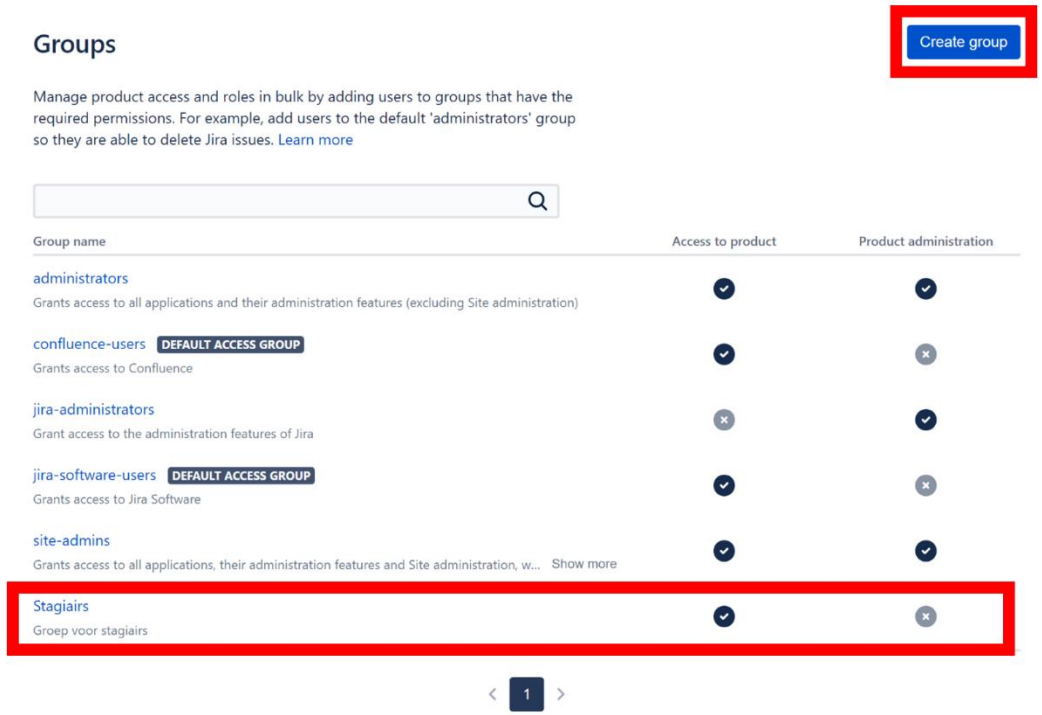
1. Groep toevoegen
2. User toevoegen aan groep
3. Jira software access toekennen aan nieuwe groep
4. Project role toevoegen
5. Permission scheme maken
6. Browse projects toekennen aan project role
7. Permission scheme toekennen aan project
8. Global permissions toekennen aan groep
9. Users verwijderen uit jira-software-users-groep
10. Browse project permissie verwijderen
11. Stagiair role toekennen aan een project

# 3 Groep toevoegen

*(Site Administration → Groups)*

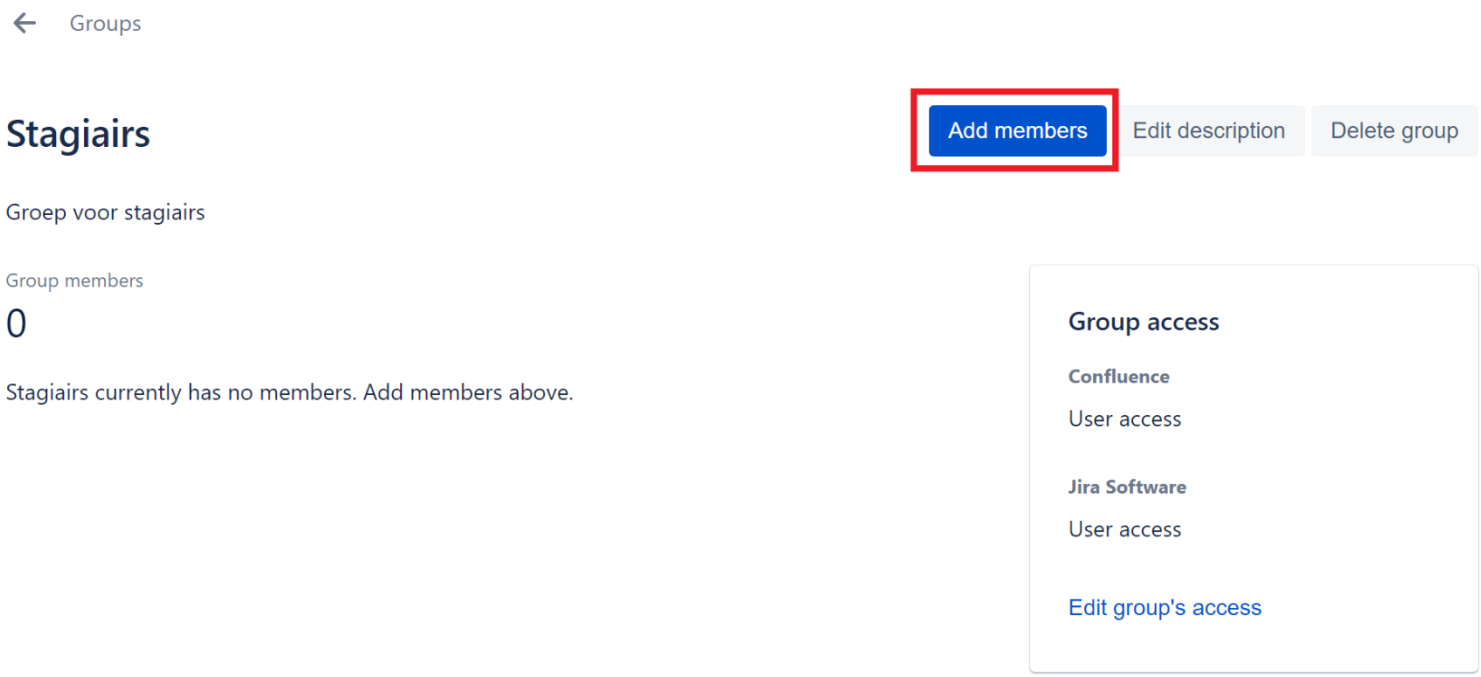
We beginnen het proces door een groep aan te maken. In dit voorbeeld willen we een groep aanmaken voor stagiairs. Dit kan enkel gedaan worden door iemand met **site-admin**-rechten.



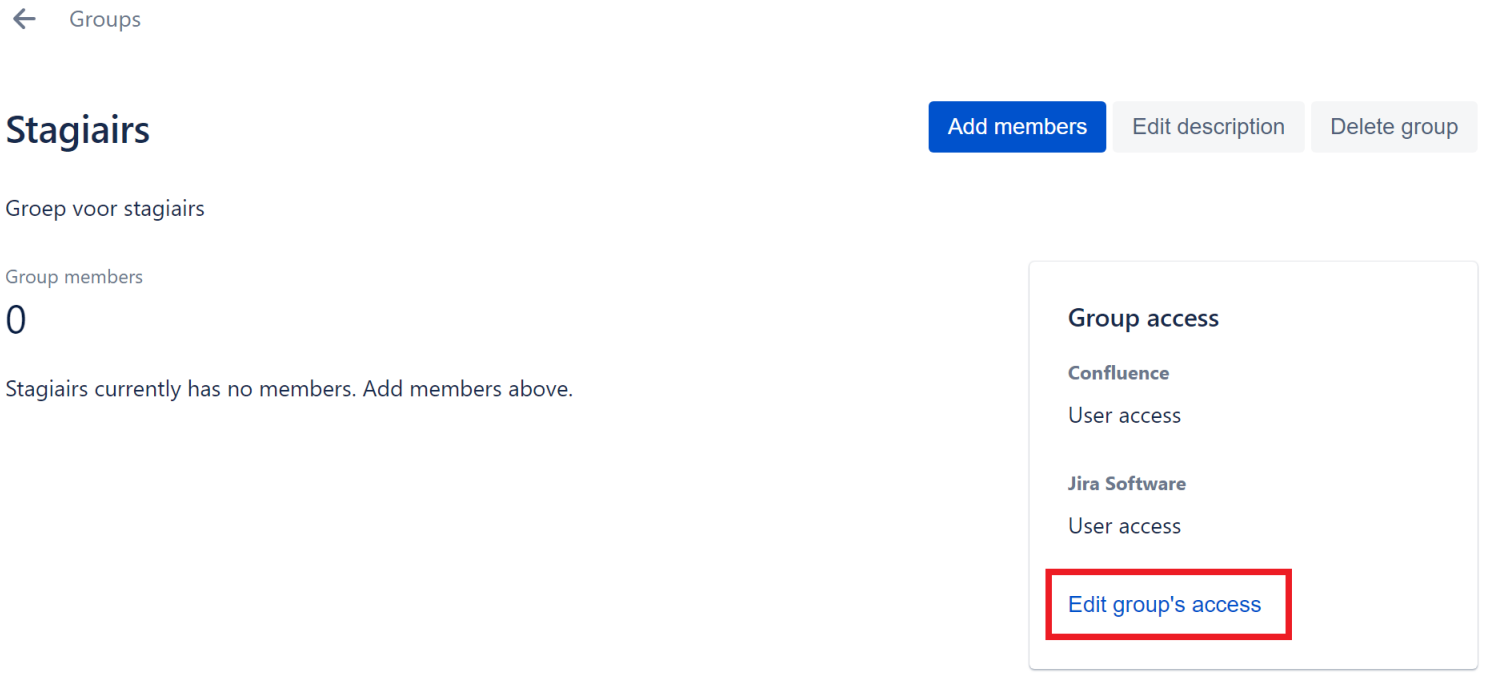
## 4 User toevoegen aan groep

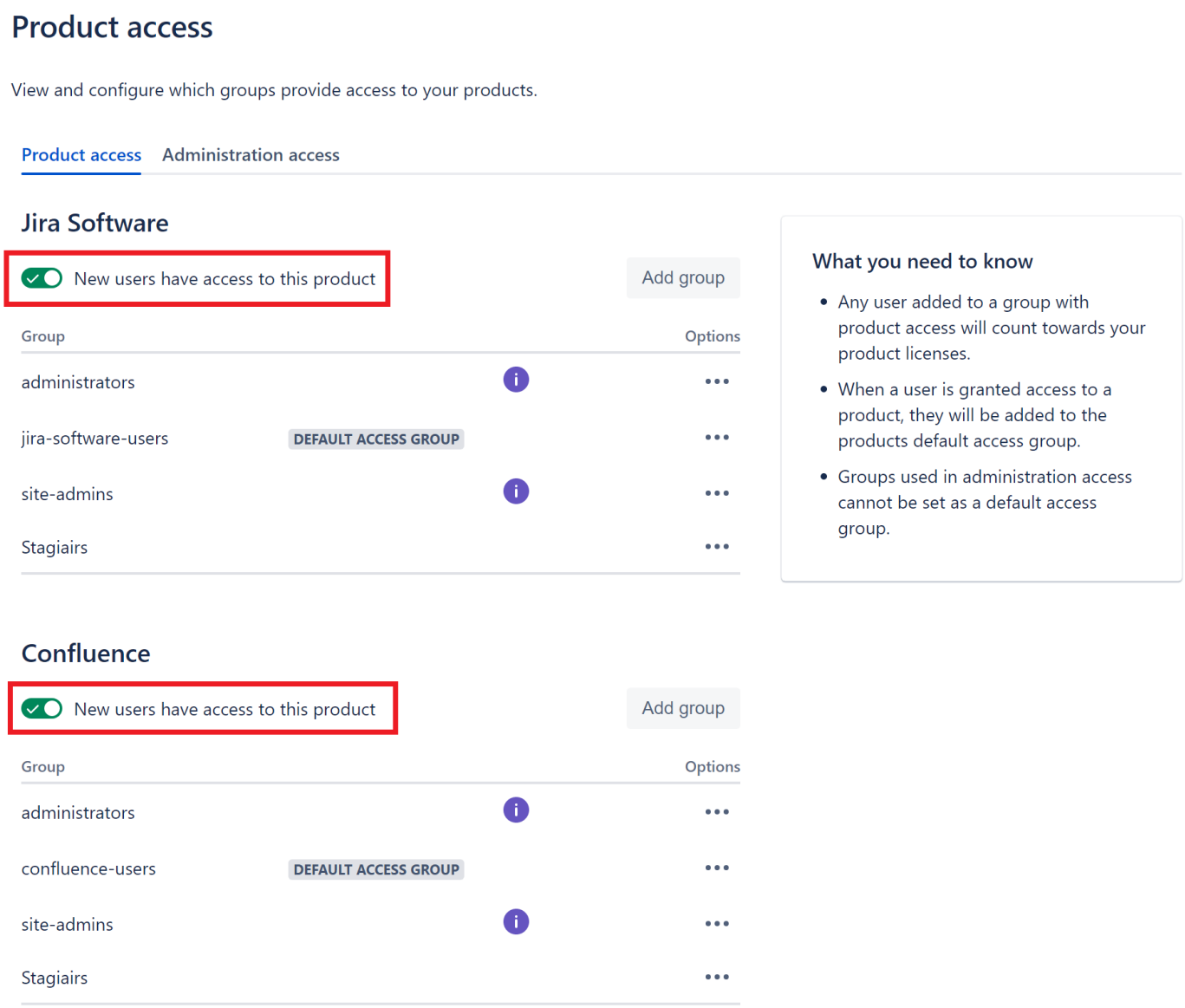
Vervolgens voegen we de gewenste gebruikers toe aan deze groep.



# 5 Jira software access toekennen aan nieuwe groep

Daarna voegen we product access toe voor deze groep. Dit zorgt ervoor dat users in deze groep zich kunnen inloggen op Jira (en eventueel Confluence)

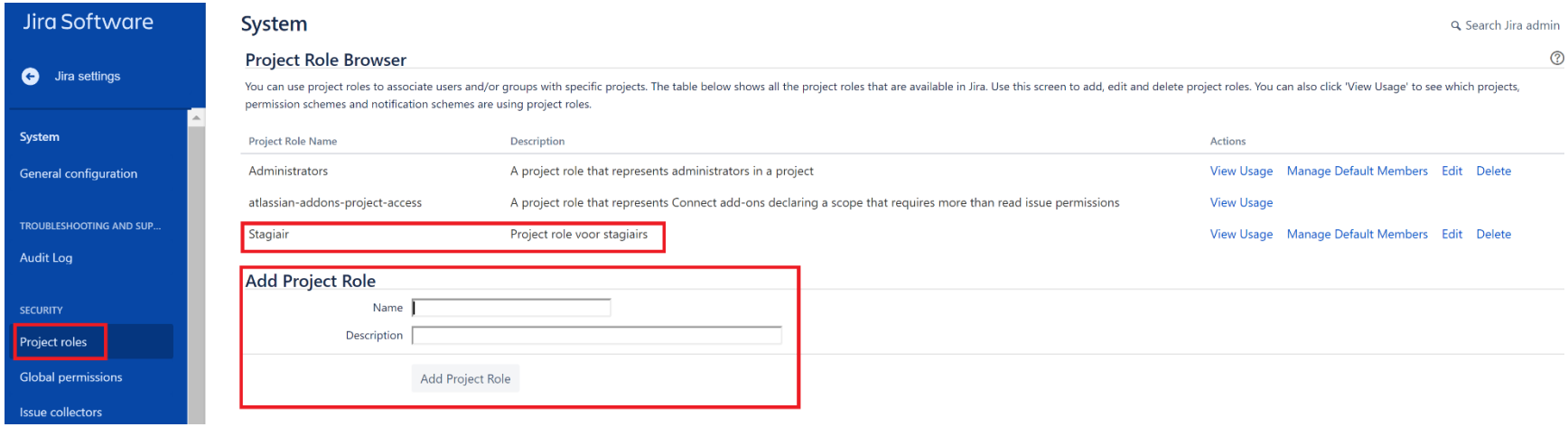



Product access
View and configure which groups provide access to your products.
Product access
Administration access
Jira Software
New users have access to this product
Add group
Group
Options
administrators
jira-software-users
DEFAULT ACCESS GROUP
site-admins
Stagiairs
What you need to know
Any user added to a group with product access will count towards your product licenses.
When a user is granted access to a product, they will be added to the products default access group.
Groups used in administration access cannot be set as a default access group.
Confluence
New users have access to this product
Add group
Group
Options
administrators
confluence-users
DEFAULT ACCESS GROUP
site-admins
Stagiairs

# 6 Project role toevoegen

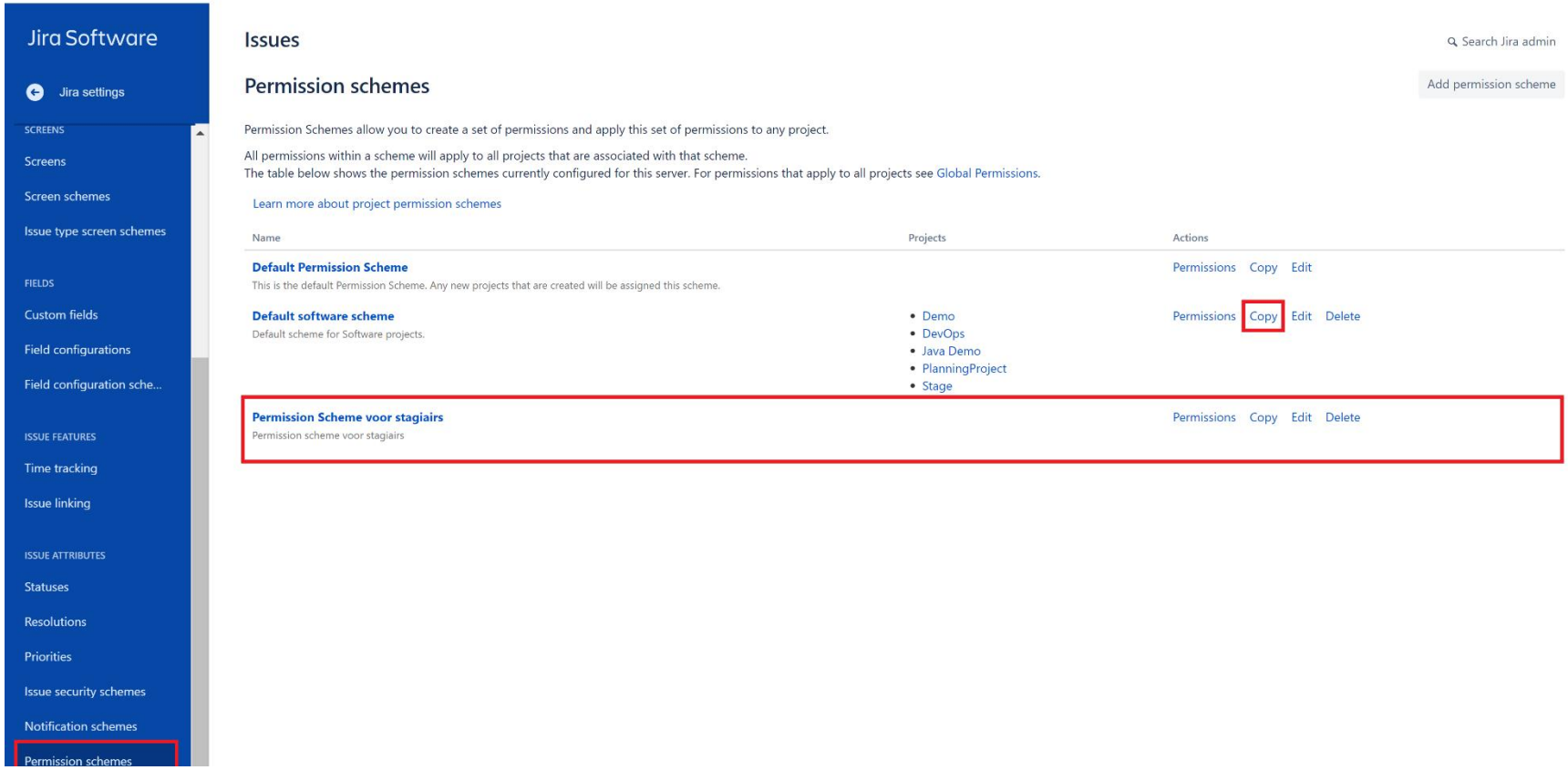
*(Jira → Jira settings → Project roles)*

Vervolgens zorgen we voor een Project Role voor onze nieuwe groep. In dit geval maken we bijvoorbeeld de rol ‘stagiair’ voor de ‘stagiairs’-groep.



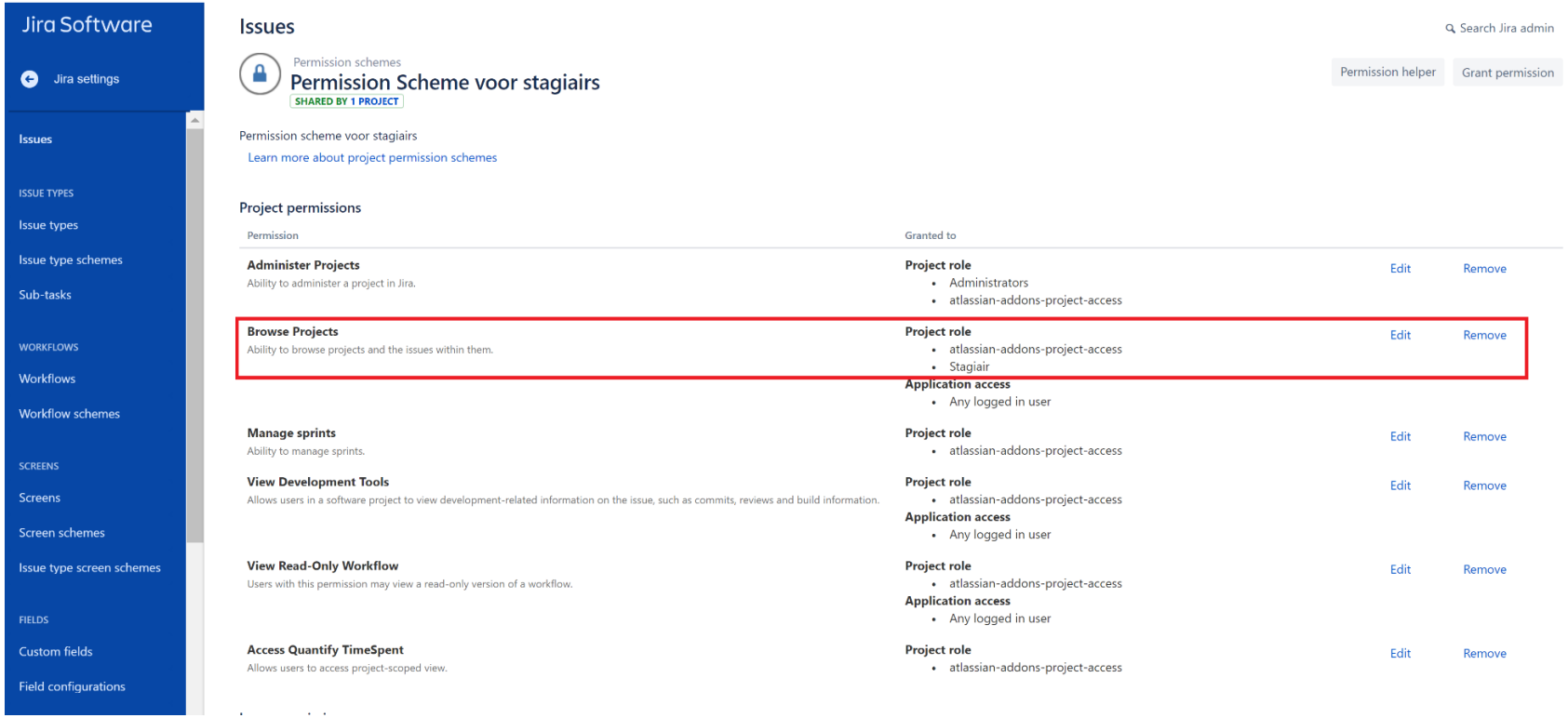
# 7 Permission scheme maken

De volgende stap is een om een permission scheme te maken voor onze nieuwe gebruikers. De gemakkelijkste manier is om gewoonweg de **default permission scheme** te kopiëren. Vervolgens kunnnen we deze kopie hernoemen en bewerken.



# 8 Browse projects toekennen aan project role

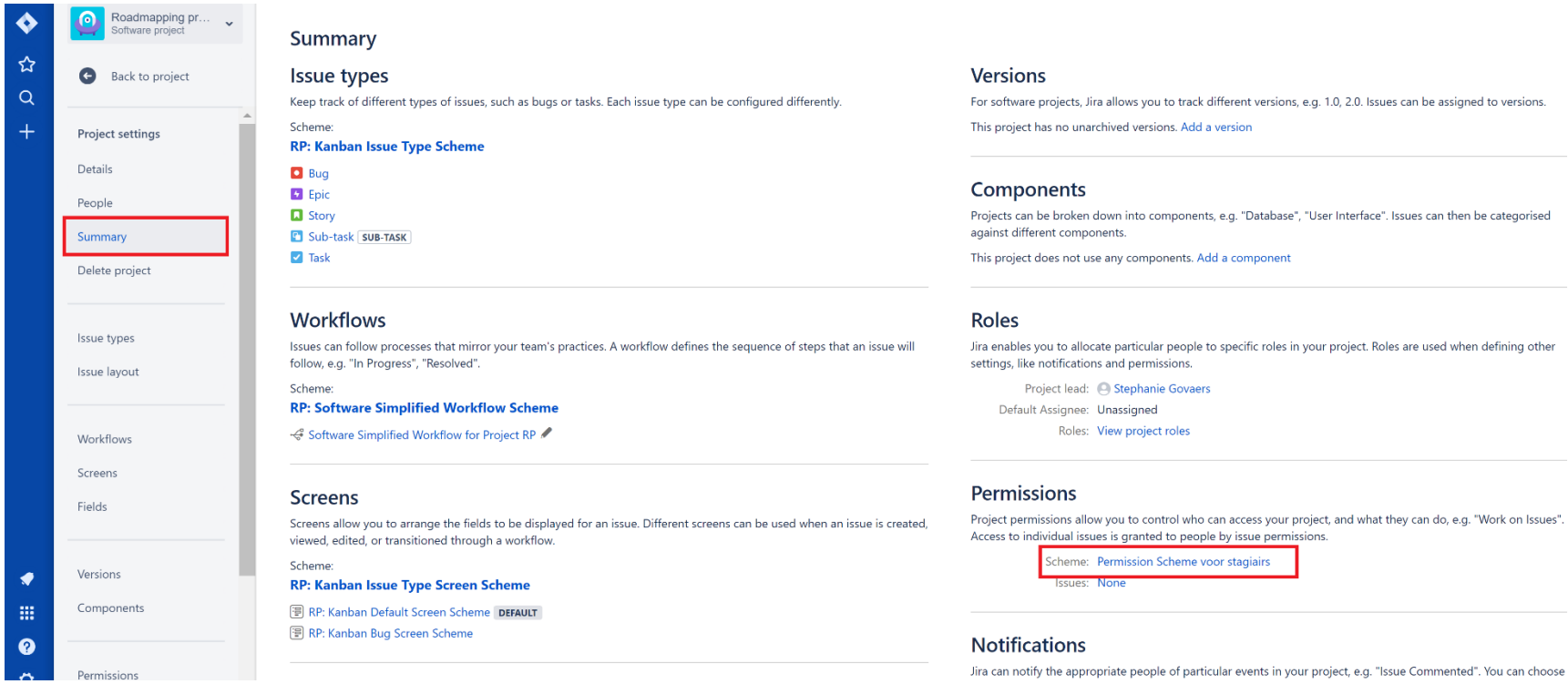
Binnen het nieuwe permission scheme moeten we aan **de browse projects permission** de project role “stagiair” toevoegen.

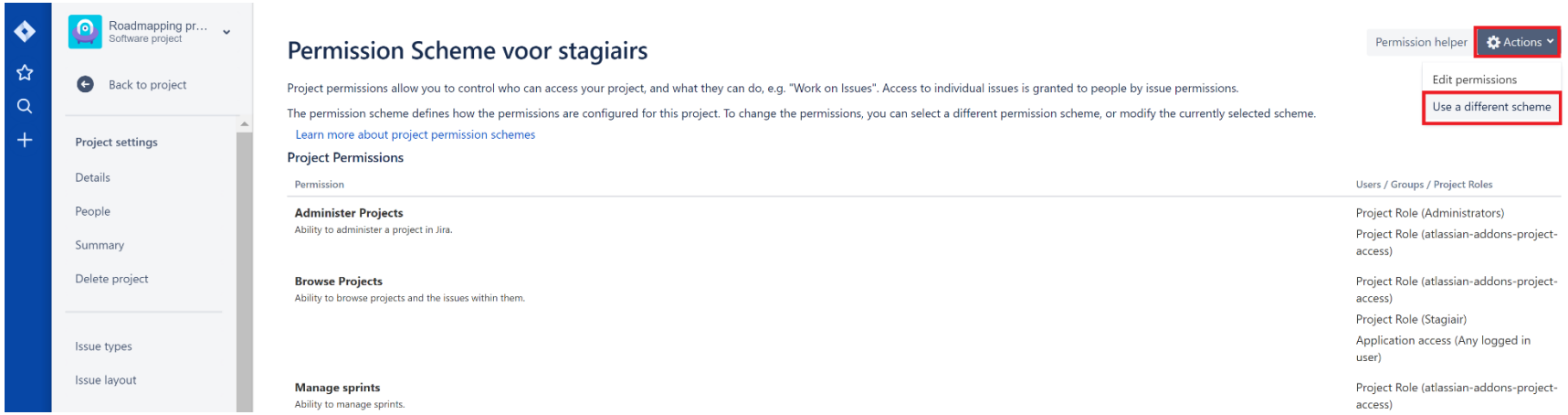


# 9 Permission scheme toekennen aan project

## (Project → Project settings → Summary → Permissions)

Vervolgens moet ons project scheme gekoppeld worden aan een of meerdere projecten waaraan we onze nieuwe users willen laten werken. Stel dat we een groep stagiairs hebben, willen we ze enkel aan hun eigen stageproject koppelen. Deze koppeling gebeurt op projectniveau.

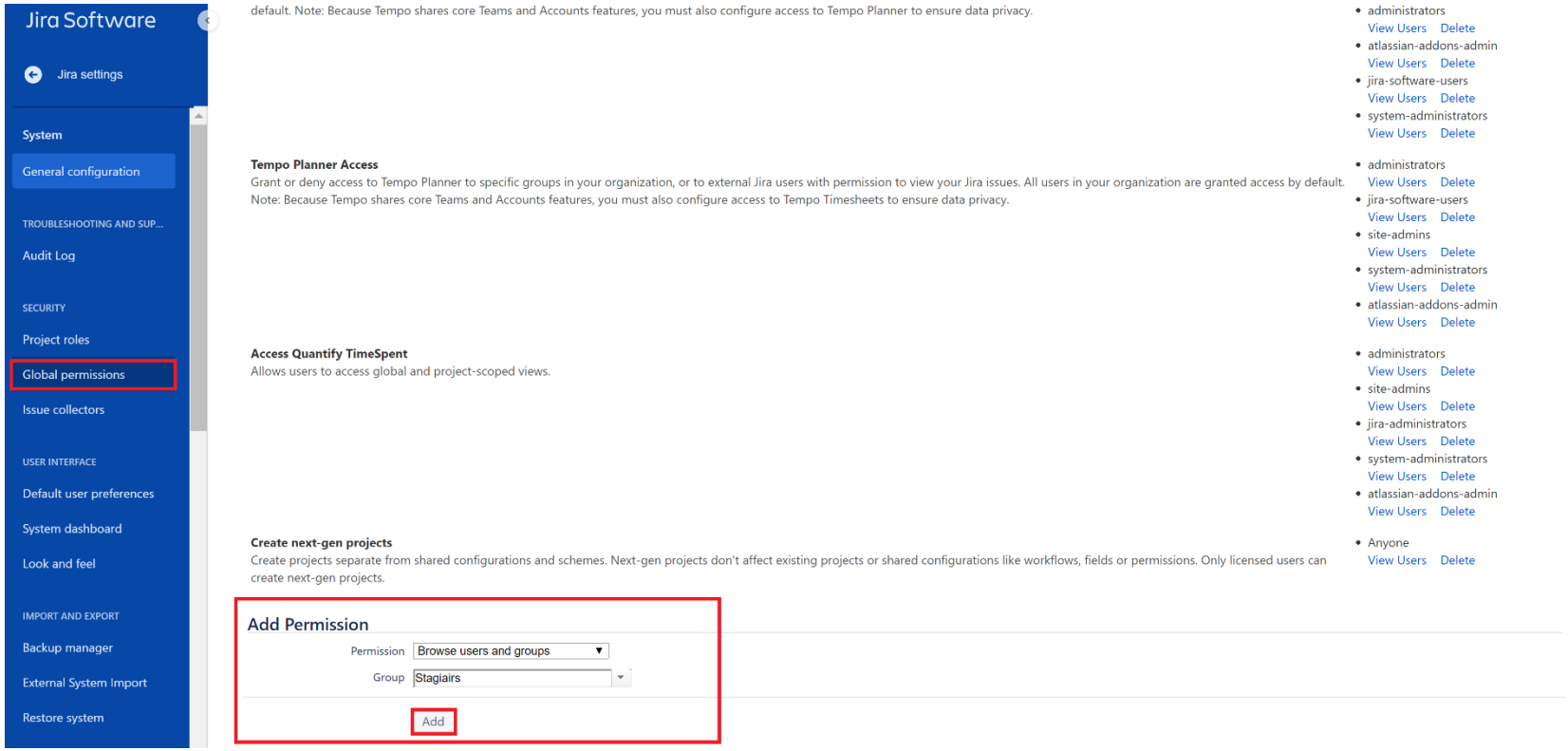



Roadmapping pr…
Software project
Back to project
Project settings
Details
People
Summary
Delete project
Issue types
Issue layout
Permission Scheme voor stagiairs
Project permissions allow you to control who can access your project, and what they can do, e.g. "Work on Issues". Access to individual issues is granted to people by issue permissions.
The permission scheme defines how the permissions are configured for this project. To change the permissions, you can select a different permission scheme, or modify the currently selected scheme.
Learn more about project permission schemes
Project Permissions
Permission helper
Actions
Edit permissions
Use a different scheme
Permission
Users / Groups / Project Roles
Administer Projects
Ability to administer a project in Jira.
Project Role (Administrators)
Project Role (atlassian-addons-project-access)
Browse Projects
Ability to browse projects and the issues within them.
Project Role (atlassian-addons-project-access)
Project Role (Stagiair)
Application access (Any logged in user)
Manage sprints
Ability to manage sprints.
Project Role (atlassian-addons-project-access)

# 10 Global permissions toekennen aan nieuwe groep

*(Jira settings → System → Global permissions)*

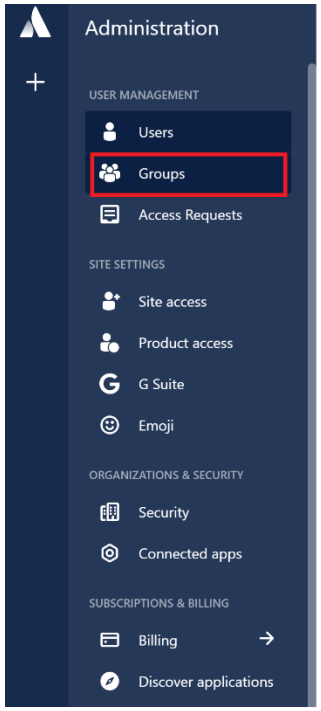
De volgende stap is om ook de nodige global permissions aan de nieuwe groep toe te kennen. We voegen de permissie **Browse users and groups** toe voor de groep "Stagiars"

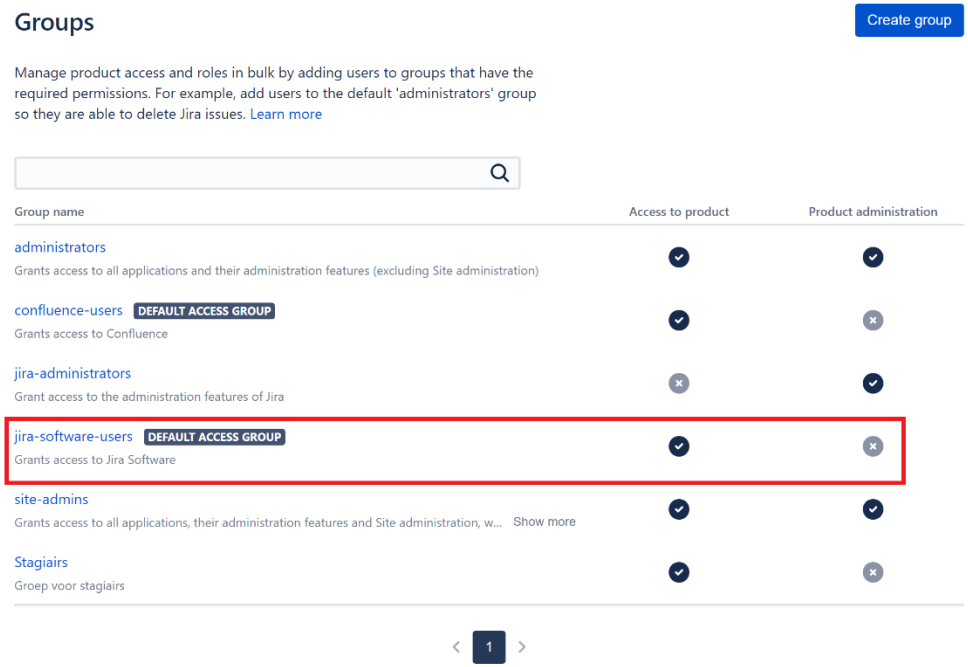


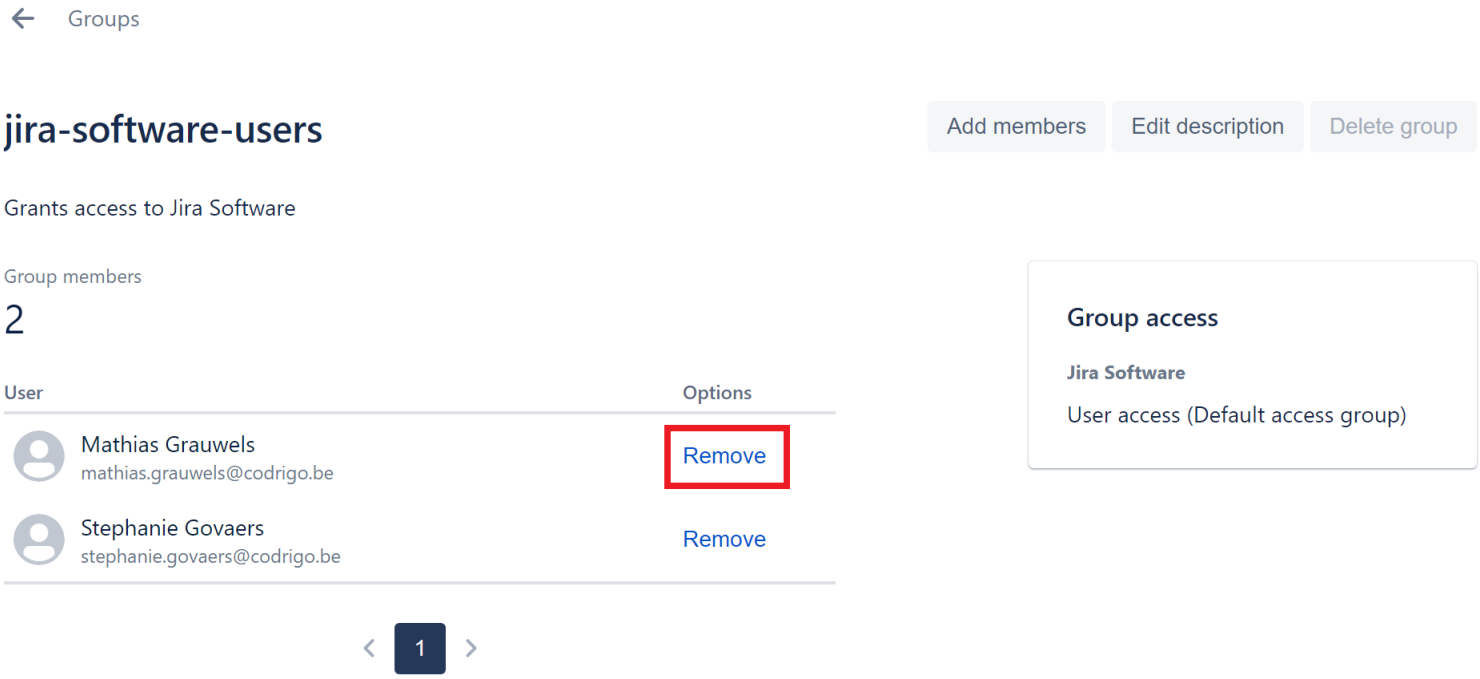
# 11 Users verwijderen uit Jira-software-users-group

*(Site administration → Groups)*

Nu moeten we de gebruikers verwijderen uit de **jira-software-users-group.** Merk op dat dit de default acces group is. Elke nieuwe gebruiker wordt automatisch in deze groep gestoken.

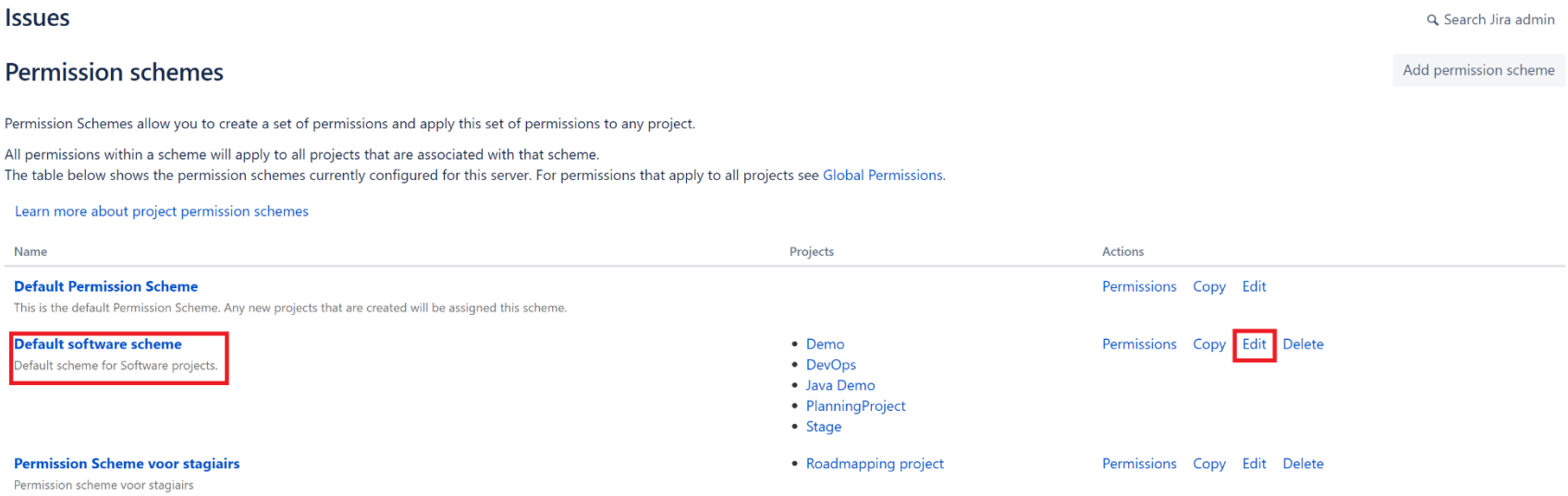


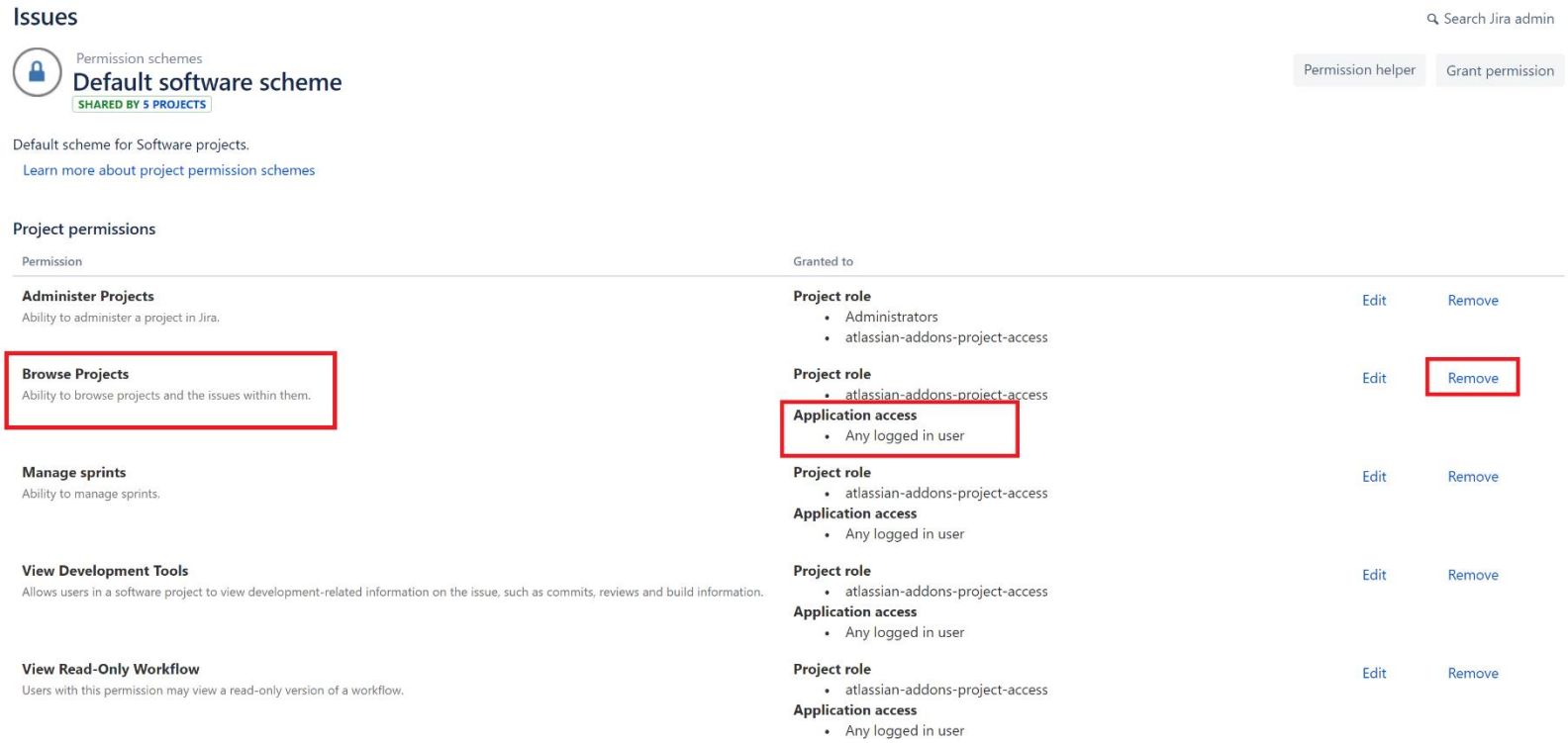




Groups
jira-software-users
Add members
Edit description
Delete group
Grants access to Jira Software
Group members
2
User
Options
Mathias Grauwels
mathias.grauwels@codrigo.be
Remove
Stephanie Govaers
stephanie.govaers@codrigo.be
Remove
1
Group access
Jira Software
User access (Default access group)

# 12 Browse project permissie verwijderen

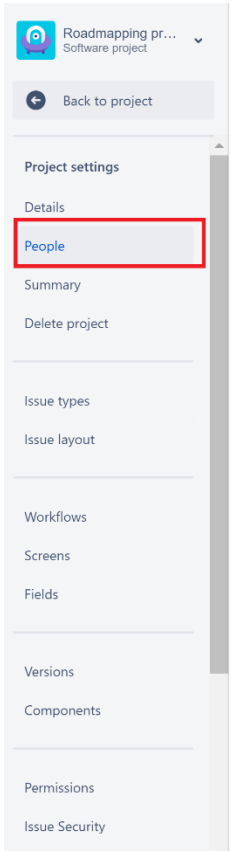
Vervolgens passen we in de **default** permission scheme de rechten van gebruikers aan. We verwijderen de toegang van **any logged in user** voor **browse projects.**

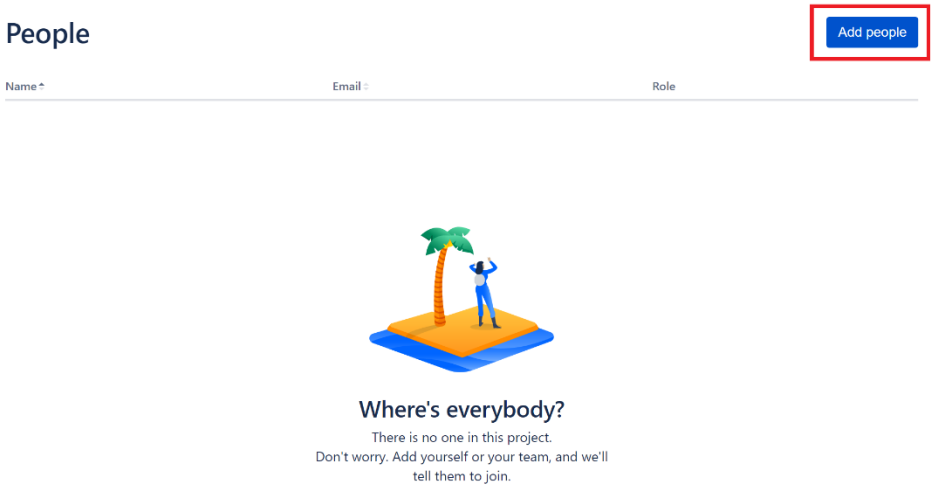


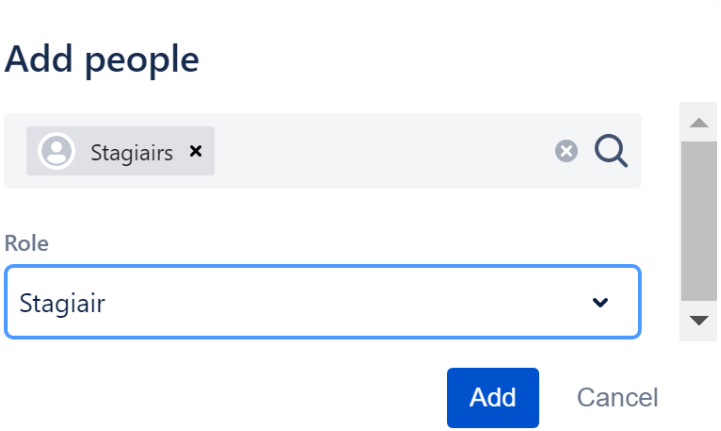

Issues
Search Jira admin
Permission schemes
Default software scheme
SHARED BY 5 PROJECTS
Permission helper
Grant permission
Default scheme for Software projects.
Learn more about project permission schemes
Project permissions
Permission
Granted to
Administer Projects
Ability to administer a project in Jira.
Project role
Administrators
atlassian-addons-project-access
Edit
Remove
Browse Projects
Ability to browse projects and the issues within them.
Project role
atlassian-addons-project-access
Application access
Any logged in user
Edit
Remove
Manage sprints
Ability to manage sprints.
Project role
atlassian-addons-project-access
Application access
Any logged in user
Edit
Remove
View Development Tools
Allows users in a software project to view development-related information on the issue, such as commits, reviews and build information.
Project role
atlassian-addons-project-access
Application access
Any logged in user
Edit
Remove
View Read-Only Workflow
Users with this permission may view a read-only version of a workflow.
Project role
atlassian-addons-project-access
Application access
Any logged in user
Edit
Remove

# 13 Stagiair role toevoegen aan een project

Tot slot voegen we op **projectniveau** de stagiar role en groep toe aan de mensen in een project.






Eens deze set-up gebeurd is, moeten nieuwe gebruikers enkel aan de nieuwe groep toegevoegd worden en uit de Jira-software-users-groep verwijderd worden. Daarnaast moet ook telkens de koppeling van de permission scheme aan nieuwe projecten gebeuren, samen met de koppeling van de groep en rol op projectniveau