Software Quality Assurance Report

Project: Software Quality Assurance Report

Project team: BBB-BC

Team: Joeri Berman (2834499)

Luc Urlings (3071081)

Stijn Baltessen (3064565)

Wouter Vandenboorn (3099571)

Floris Feddema (3188256)

Davy de Haas (3099695)

Opdrachtgever: Jeffrey Cornelissen

Version: 0.2

Version Date: 17 maart 2020

Status: Concept

# Inleiding

# Inhoudsopgave

[**Inleiding**](#_a7rcsa9lg34k) **2**

[**Inhoudsopgave**](#_2n0sw24u6pdp) **3**

[**Automatische acceptatie tests**](#_en8t1seh52hy) **4**

[**Test tooling (Long List)**](#_67xe0jx77isp) **5**

[UI Testing](#_n5wmye3obuuv) 5

[Backend Testing](#_q6l53g84r0s9) 6

[JavaScript](#_e7d9dxoym3v6) 6

[C#](#_atlpp4famghy) 6

[Java](#_lcihkzwosw2j) 6

[**Tooling met potentie**](#_gaefe5ake97k) **7**

[Huidige situatie](#_yn4p24898ce) 7

[Verdere eisen van testen vanuit de PO](#_tluraj7o2wg) 8

[Browser based](#_hqbf38j3r9x4) 8

[Rapport generatie](#_nns6c1l8qud8) 8

[Opname](#_aq64pnggdh8z) 8

[Verdere eisen van testen vanuit de Developers](#_l6oxl33qsrpm) 8

[Integratie met tooling](#_85l78gxfkgoq) 8

[Support meerdere talen](#_zhdl3to4n8e1) 8

[Industriële standaard](#_q9tnh52r3r7y) 8

[Ervaring team](#_2aotrqhykorc) 9

[Gradual learning curve](#_29upbldyo34i) 9

[Actieve Developers](#_w1jhf3rsa1d0) 9

[Selectie matrix](#_px80fx4m6axe) 9

[Verdere uitwerking PoC’s](#_t8pxgila5stq) 12

[**User story acceptatie**](#_vpmmwogm4w9b) **13**

[Hoe willen wij de ‘definition of done’ inrichten met automatisch testen?](#_e98p6alfh57v) 13

[Schets van pipeline met automatisch test tooling](#_223tt6965pmc) 15

[**PoC bevindingen**](#_hmiow3rris5i) **16**

[**Conclusie & advies aan de klant**](#_cd2x055bed1) **18**

# 

# Automatische acceptatie tests

De acceptatietest is een test waar het gehele systeem wordt getest. Het doel van de test is om het systeem te testen met de business requirements en de eisen van de gebruiker. Hier wordt ook bepaald of het voldoet aan de eisen voor een oplevering.

De test is de laatste stap voordat een versie van de software gereleased wordt, de omgeving waarin dit plaatsvindt is meestal zo realistisch mogelijk. Acceptatie tests zijn anders dan bijvoorbeeld unit tests en integration tests. Bij een acceptatietest wordt er op knoppen gedrukt, op links gedrukt en texten ingevuld. Hier wordt het gebruik van de software zo realistisch mogelijk nagedaan.

Wanneer deze tests handmatig moeten worden gedaan gaat hier veel tijd in zitten, ook zijn dit taken waar doorgaans weinig motivatie voor zijn. Door een tool te gebruiken die deze tests automatisch doet kan deze tijd nuttig worden besteed aan ontwikkelen. Wanneer er nieuwe features uitkomen zullen hier ook tests voor moeten worden geschreven, deze kunnen mee worden genomen in de DoD van een story. Dit zorgt ervoor dat deze altijd gemaakt worden en slagen. Door alleen een releases te doen wanneer alle acceptatietesten slagen kan men zeker zijn dat alle features werken zoals ze bedoeld zijn. Er zijn veel tools beschikbaar om dit te doen. Hier zal een keuze voor kunnen worden gemaakt afhankelijk van de technologieën die binnen het project zijn gebruikt.

# 

# Test tooling (Long List)

## UI Testing

* **Ranorex**

Desktop, web & mobiel testing scripted van recorded testing

* **TestComplete - Desktop, Web & Mobile**

Recorded testing tool voor desktop, web en mobiel

* **QMetry Automation Studio**

Integrated testing voor Eclipse IDE

* **LEAPWORK**

Cross applicatie testing

* **Katalon Studio**

Integratie met Jira, Git, Slack en andere. Werkt op Windows, Linux, en macOS.

* **Test Sigma**

AI-driven test tool.

* **LambdaTest**

Browser testing tool, integratie met Jira, Git en Slack.

* **Qualibrate**

Browser en SAP testing tool.

* **Worksoft**

Industrie standaard voor testen van SAP tools.

* **CrossBrowserTesting**

Voor testen van Selenium tests op verschillende (mobiele) browsers

* **Testimony**

Testing tool voor SAP.

* **ZeuZ Test Automation Framework**

Testing framework voor mobiel, cloud, web en API’s. Werkt goed samen met CI/CD.

* **Selenium**

Grootste web testing framework. Werkt op verschillende besturingssystemen.

* **Subject7**

Integreerbaar CI/CD. Kan databases, web services, desktop en mobiele apps testen.

* **Appium**

Open source test tool voor mobiel.

* **Micro Focus UFT**

Cross Platform browser testing tool van Hewlett-Packard

* **Telerik Test Studio**

Recorded testing tool voor C#, met name GUI en API testing

* **IBM Rational Functional Tester**

Scripted regressie en integratie testing tool.

* **Coded UI Test**

Een UI test framework in Visual Studio Premium.

## Backend Testing

Hierbij zullen we vooral gaan kijken naar JS omdat we dit gaan gebruiken in het groepsproject. Daarnaast zal er ook gekeken worden naar Java en .NET omdat dit platformen zijn die we waarschijnlijk ook gaan gebruiken.

## JavaScript

* **MochaJS**

Test framework voor frontend en backend testing

* **JESTJS**

Unit testing framework

* **Jasmine**

Testing framework

* **Enzyme**

Testing utility voor een react applicatie

## C#

* **NUnit**

Unit testing framework

* **xUnit.net**

Unit testing framework

## Java

* **JUnit**

Unit testing framework

* **NUnit**

Unit testing framework

# Tooling met potentie

In dit hoofdstuk wordt besproken waarom 2 tooling het meeste potentie hebben en waarom wij ervoor kiezen om deze tooling verder uitwerken in een PoC voor de PO.

De huidige situatie wordt uitgelegd, verdere eisen worden vastgesteld en uiteindelijk wordt een matrix gebruikt om alle eisen af te wegen met de beschikbare tooling die in de long list staan.

Ook wordt het belang van de PoC’s benadrukt, aangezien dit resources kost willen we het belang ervan benadrukken.

## Huidige situatie

Om te redenering te snappen waarom wij voor deze tooling kiezen moeten we eerst vaststellen hoe nu het testen wordt gedaan.

Op dit moment worden features die af zijn op een testomgeving gezet waar de PO zelf handmatig de features test volgens acceptatie criteria. De PO vindt het heel belangrijk dat de applicatie op de UI/front end werkt. Hierom is het heel belangrijk dat wij user acceptatie criteria kunnen gebruiken in onze test tooling.

Dit betekent dat wij voor een testing tool gaan waarmee we een rapport kunnen maken aan de hand van testen die op de frontend zijn gebeurt. Denk hierbij als voorbeeld de volgende user acceptatiecriteria:

|  |
| --- |
| Stap 1: gebruiker drukt op knop ‘login’.  Stap 2: gebruiker krijgt een melding met daarin ‘U moet een gebruikersnaam en wachtwoord invoeren om in te loggen.’ |

De PO vind het belangrijk dat wij onze applicatie juist inrichten. Echter maakt het hem niet veel uit hoe onze backend eruit ziet. Hij wil dat de front end, “doet wat het moet doen”. En de eisen van wat het moet doen kan de PO duidelijk uitleggen, hierdoor kunnen wij duidelijk gebruiker testcriteria opzetten.

Deze test criteria kunnen we laten goedkeuren door de PO, dit biedt ons de mogelijkheid aan om volgens zijn eisen een automatische test omgeving op te zetten.

Dus hiermee weten we dat een eis van de test tooling is dat het de voorkant kan doorlopen net zoals een gebruiker. Dit creëert het veld “Mogelijk om UI tests net zoals een gebruiker uit te voeren” in onze matrix. Hiermee kunnen we dus van onze long list alle C# and java test toolings wegstrepen.

## Verdere eisen van testen vanuit de PO

Om verder andere tooling te kunnen wegstrepen moeten we andere eisen van ons gewilde tooling vast stellen. En om uiteindelijke de matrix juist aan te kunnen leveren.

### Browser based

Het product dat wij maken is alleen web-based dus de keuze om dan een tool te pakken dat gespecialiseerd in browser based testing. Dit creëert het veld “Gespecialiseerd zijn in browser based testing” voor onze matrix

### Rapport generatie

Onze PO wil op het einde van alle test een duidelijk overzicht krijgen wat werkt en wat geaccepteerd is. Dus onze tooling moet de mogelijkheid bieden om een rapport zoals dit te genereren. Dit creëert het veld “Rapport kunnen genereren”.

Als extra argumentatie voor onze developers is een rapport ook overzichtelijk om gefaalde tests makkelijk te plaatsen in onze applicatie.

### Opname

De Product Owner wilt graag verzekerd zijn dat alle tests juist zijn uitgevoerd. Hij wilt dus de mogelijkheid krijgen dat hij de mogelijkheid heeft een opname terug te zien van de tests die uitgevoerd zijn door de tooling. Dit creëert het veld “Opname van test terugzien”.

## Verdere eisen van testen vanuit de Developers

### Integratie met tooling

Als team werken wij met Jira Cloud wij willen hierin als kunnen integreren. Dit creëert het veld “Integreerbaar in Jira Cloud”.

### Support meerdere talen

We willen deze test tooling met meerdere talen kan werken. Voor het ontwikkel team is het belangrijk dat wanneer wij een andere taal implementeren in onze front-end dat dit ook met dezelfde test tooling kan worden getest. Dit creëert het veld “tooling support industrieel standaard talen”.

### Industriële standaard

Om onze kwaliteit van onze test tooling te kunnen verdedigen is het ook belangrijk om te weten wat de industriële standaard is. Hierdoor kunnen we de legitimiteit van de testen beargumenteren. Dit creëert het veld “val tooling binnen industriële standaard”.

### Ervaring team

Wanneer een persoon van het dev-team ergens al ervaring mee heeft kan dit veel tijd besparen voor het team omdat we dan een centraal persoon hebben die kan dienen als informatie/ instructie bron. Dit creëert het veld “Is er iemand in het team die hier al ervaring mee heeft”.

### Gradual learning curve

Wanneer een tool makkelijk op te pakken is betekent dit een gradual learning curve. Voor een team betekent dit weer tijd besparing wat belangrijk is voor smoother development van nieuwe features. Dit creëert het veld “Gradual learning curve”.

### Actieve Developers

We willen niet alleen dat de tooling binnen industriële standaarden valt, we willen ook dat de tooling actief gebruikt wordt door andere developers en dat hier een community van is. Dit zorgt ervoor dat wanneer er eventuele vragen voorkomen -- die het team niet kan beantwoorden -- wel beantwoord kunnen worden door externe bronnen. Dit creëert het veld “Actieve community”.

## Selectie matrix

Matrix waarin we de eisen van vorige stukken zetten in vergelijking met alle tooling van de long list.

We halen de javascript, c# en java stukken van de long list weg en de gehele eis “Mogelijk om UI tests net zoals een gebruiker uit te voeren” deze 2 strepen elkaar weg en hierdoor krijgen we meer overzicht.

We hopen uit deze selectie matrix minder dan 4 tooling over te houden omdat dit tijd bij de PoC’s gaat besparen.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Gespecialiseerd in browser based testing | Rapport kunnen genereren | Opname van test terugzien | Integreerbaar in Jira cloud | Tooling valt binnen industriële standaard | Is er iemand in het team die hier al ervaring mee heeft | gradual learning curve | actieve community |
| Ranorex | FALSE | TRUE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE |
| TestComplete - Desktop, Web & Mobile | FALSE | TRUE | TRUE | FALSE | FALSE | FALSE | FALSE | TRUE |
| Qmetry Automation Studio | TRUE | TRUE | FALSE | TRUE | TRUE | FALSE | FALSE | FALSE |
| LEAPWORK | FALSE | TRUE | TRUE | FALSE | FALSE | FALSE | TRUE | FALSE |
| Katalon Studio | TRUE | TRUE | TRUE | TRUE | TRUE | TRUE | TRUE | TRUE |
| Test Sigma | TRUE | TRUE | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE |
| LambdaTest | TRUE | TRUE | TRUE | TRUE | TRUE | FALSE | TRUE | FALSE |
| Qualibrate | TRUE | TRUE | FALSE | FALSE | TRUE | FALSE | TRUE | FALSE |
| Worksoft | TRUE | TRUE | FALSE | FALSE | TRUE | FALSE | TRUE | FALSE |
| CrossBrowserTesting | TRUE | TRUE | TRUE | TRUE | TRUE | FALSE | TRUE | FALSE |
| Testimony | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| ZeuZ Test Automation Framework | TRUE | TRUE | FALSE | TRUE | TRUE | FALSE | TRUE | FALSE |
| Selenium | TRUE | TRUE | TRUE | TRUE | TRUE | TRUE | TRUE | TRUE |
| Subject 7 | TRUE | TRUE | FALSE | FALSE | TRUE | FALSE | TRUE | FALSE |
| Appium | TRUE | TRUE | TRUE | FALSE | TRUE | FALSE | TRUE | TRUE |
| Micro Focus UFT | TRUE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| Telerik Test Studio | TRUE | TRUE | TRUE | FALSE | TRUE | FALSE | FALSE | TRUE |
| IBM Rational Functional Tester | TRUE | TRUE | TRUE | FALSE | TRUE | FALSE | TRUE | FALSE |
| Coded UI Test | TRUE | TRUE | TRUE | FALSE | TRUE | FALSE | TRUE | FALSE |

Figuur test matrix

|  |
| --- |
| Diagram |
| Figuur matrix resultaten overzicht.  Uit de matrix blijkt dat er 2 toolings zijn die aan 8 van de 8 eisen voldoen, **Katalon** en **Selenium**. Aangezien we weinig tijd hebben is het belangrijk om tijd te besparen en dus alleen deze 2 verdere uitwerken in een PoC.  Zal niet verder worden gekeken naar de 3 resultaten die aan 6 van de 8 eisen voldoen. Echter als de PO niet tevreden is over de tooling die nu wordt uitgewerkt aangezien deze niet voldoen aan zijn eisen.  Dus in dit geval zal er nog worden gekeken naar de tooling die 6 van de 8 eisen heeft behaald, aangezien een van de punten die bij elk niet behaald is “ervaring binnen het team” is. Waardoor deze tooling nooit meer dan 7 hadden kunnen halen. Dit maakt het verschil tussen de top 2 en de 3 die daarop volgen een stuk kleiner als je de kolom “ervaring binnen het team” weggehaald.  De uiteindelijke keuze is gevallen om **Katalon** en **Selenium** in de PoC uit te werken. |

## 

## Verdere uitwerking PoC’s

In dit hoofdstuk hebben we 2 tooling gekozen die potentie hebben om ons product volgens onze en de PO’s wensen te voltooien. Deze 2 tooling gaan we uitproberen door een Proof of Concept per gekozen tooling te gebruiken.

Uit deze PoC’s willen we een conclusie halen welke de meeste voordelen voor onze situatie heeft, deze willen we dan als voorstel aan de PO brengen met de oplossing die verder besproken is in het hoofdstuk: “User story acceptatie”.

# User story acceptatie

In dit hoofdstuk wordt besproken over hoe de acceptatie van user stories in kader van automatisch testing geaccepteerd worden. Hierbij denken we aan de afspraken die hierbij komen met de PO. In dit hoofdstuk worden de volgende vraagstukken beantwoord en besproken.

De vraag: ”Hoe willen wij de *‘definition of done’* inrichten met automatisch testen?” beantwoorden en bespreekbaar maken met de PO.

De vraag: “Hoe ziet automatisch testen eruit in de CI/CD pipeline?” wordt ook beantwoord door een schets.

## Hoe willen wij de *‘definition of done’* inrichten met automatisch testen?

Op dit moment is de definition of done die afgesproken is met de PO als volgt:

|  |
| --- |
| “Stap 1: Een feature wordt ontwikkelt op een feature branch wanneer die feature af is en geen bugs inhoudt, zal dit naar de test-feature branch worden gepusht.  Stap 2: Hierbij is het nu aan de PO om de feature manueel te testen op de test-feature branch, de feature heeft user acceptance criteria waar het aan moet voldoen. Deze criteria gebruikt de PO om het te testen.  Stap 3: Zodra de PO vind dat de feature aan de acceptance criteria voldoet laat hij dit het dev-team weten.  Stap 4: Nu is het aan het dev-team om de master te updaten van de feature-test branch om de feature live te zetten. Is de master succesvol geüpdatet en gelauncht en staan de verandering live, dan is de story per definitie done.” |

De stappen die essentieel zijn om te veranderen voor het realiseren van het automatisch testen zijn stap 2 en 3. Hoe wij deze stappen willen veranderen is door een van onze twee geschikt tooling die in het hoofdstuk “Selectie matrix” staan toe te passen in stap 2 en 3.

Deze toepassing zal er dan als volgt uitzien:

|  |
| --- |
| Stap 1: Een feature wordt ontwikkelt op een feature branch. Wanneer die feature af is en geen bugs inhoudt, zal dit naar de test-feature branch worden gepusht. |
| Stap 2 : De automatische test tooling krijgt een melding dat een new feature op de test-feature branch staat en dat deze getest zal moeten worden volgens acceptatie criteria.  Stap 3: Zodra de automatische test tooling dit juist vindt krijgt het dev-team een melding. |
| Stap 4: Nu is het aan het dev-team om de master te updaten van de feature-test branche om de feature live te zetten. Is de master succesvol geupdate en gelauncht en staan de verandering live, dan is de story per definitie done.” |

Wat wij willen doen in kader om dit bespreekbaar te maken met de PO is door nog een stap op te zetten voor stap 2 en dat is de criteria van de test samen door te nemen zodat de PO het mee eens is dat de uitkomst klopt.

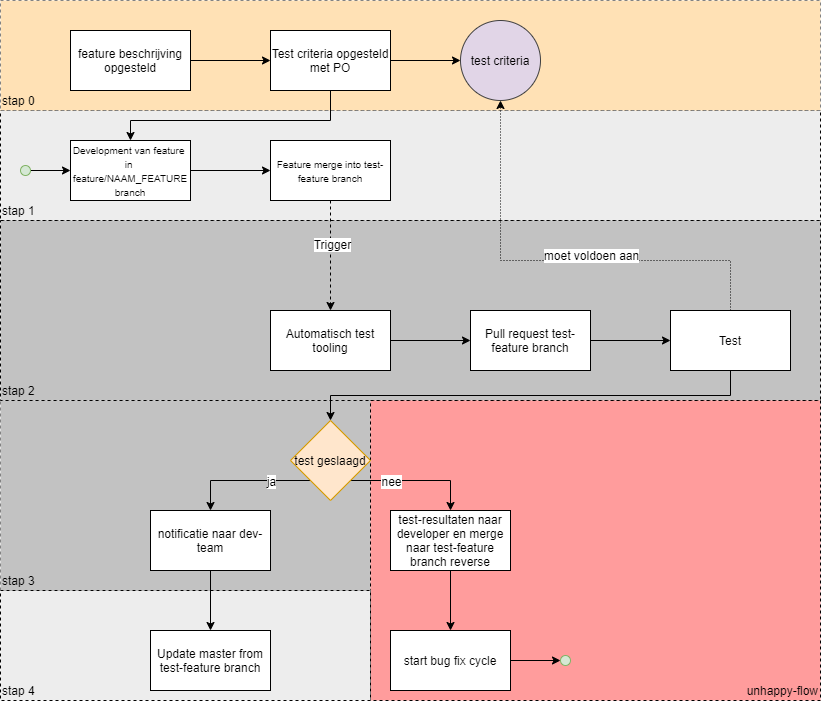
Deze extra toepassing zal er als volgt uitzien.

|  |
| --- |
| Stap 0: De developer die de feature zal ontwikkelen gaat met de PO bespreken wat de acceptatiecriteria zijn voor de automatisch test tooling, die bij stap 2 worden gevolgd. |
| Stap 1: Een feature wordt ontwikkelt op een feature branch wanneer die feature af is en geen bugs inhoudt, zal dit naar de test-feature branch worden gepusht. |
| Stap 2 : De automatische test tooling krijgt een melding dat een new feature op de test-feature branch staat en dat deze getest zal moeten worden volgens acceptatiecriteria.  Stap 3: Zodra de automatische test tooling dit juist vindt krijgt het dev-team een melding. |
| Stap 4: Nu is het aan het dev-team om de master te updaten van de feature-test branch om de feature live te zetten. Is de master succesvol geupdate en gelauncht en staan de verandering live, dan is de story per definitie done.” |

Wij hopen hiermee de PO het vertrouwen te geven dat hij de automatisch test tooling uitkomst kan vertrouwen omdat hij deel heeft genomen aan het opstellen van de acceptatiecriteria die de test tooling zal gebruiken.

## 

## Schets van pipeline met automatisch test tooling



# PoC bevindingen

We hebben gekozen voor de automatische testtools Selenium en Katalon. Voor beide projecten is een kleine PoC opgezet waarin vooral gekeken is naar het gemak van het opzetten van een automatische UI test.

De uitgewerkte PoC zijn te vinden binnen onze GitHub organisatie :

* Selenium: <https://github.com/S65-2-project/Case_Study_Selenium>
* Katalon: <https://github.com/S65-2-project/Case_Study_Katalon>

Voor beide projecten hebben we de tooling laten navigeren naar <https://fhict.nl>. Op deze locatie wordt vervolgens automatisch login gegevens ingevuld in de daarvoor bestemde tekstvelden. Nadat deze zijn ingevuld wordt er op de login knop gedrukt. Aangezien we niet valide gegevens invullen is het de bedoeling dat de login poging mislukt. De website toont daarop een error melding. Wanneer deze wordt gedetecteerd door de testing software moet de test slagen. Als deze melding niet verschijnt moet de test falen.

**Gebruikersgemak**

Beide projecten waren relatief snel en gemakkelijk op te zetten. Tijdens het opzetten van de Selenium omgeving hadden we enkele problemen maar dat lag aan Java en niet aan Selenium. Katalon was over het algemeen net iets sneller met het opzetten.

Bij beide tools is het mogelijk om alle handelingen op te nemen. Dit maakt het erg makkelijk om de acceptance test op te zetten.

**Leesbaarheid**

Het is natuurlijk gedeeltelijk ervaring en gedeeltelijk persoonlijke voorkeur maar het is natuurlijk wel een belangrijk onderdeel om mee te nemen binnen de conclusie. Hoe snel kan er gezien worden wat er gedaan wordt door alleen maar naar de code te kijken?

Selenium

driver.get("https://fhict.nl/");

driver.findElement(By.*xpath*("//\*[@id=\"userNameInput\"]")).sendKeys("henk@fhict.nl");

Katalon

WebUI.navigateToUrl('https://fhict.nl/')

WebUI.setText(findTestObject('Object Repository/Page\_Sign In/input\_Sign in with your organizational acco\_d4c899'), 'henk@fhict.nl')

Beide voorbeelden hierboven voeren dezelfde actie uit. Er wordt genavigeerd naar een pagina en hier wordt vervolgens een e-mailadres ingevuld in het login veld.

Een verschil tussen de tools bij de huidige manier van implementatie is het vinden van een element. Selenium gebruikt hier Xpath voor, Katalon kan dit ook, maar in deze PoC wordt het net iets anders gedaan. Katalon indexeert de pagina.

Katalon slaat de xpath gegevens van een element op in een folder binnen het project. Hierin staan alle gegevens van een element. Vervolgens kan er gerefereerd worden aan dit element door middel van de object referentie die is opgeslagen. Hierdoor is er geen losse xpath nodig binnen het schrijven van het testscript.

**Automatic web browser latency prevention**

Een voordeel voor Katalon is dat er binnen deze tool een automatisch systeem ingebouwd is die wacht op het laden van een pagina. Er zullen dus geen acties worden uitgevoerd totdat een pagina volledig geladen en dus volledig interactief is.

Binnen Selenium is dit niet automatisch een feature, developers kunnen handmatig een wait toevoegen die wacht totdat de pagina geladen is maar dit gebeurt niet automatisch.

**Documentatie**

Beide tools hebben goede documentatie bronnen die online te vinden zijn.

* [The Selenium Browser Automation Project :: Documentation for Selenium](https://www.selenium.dev/documentation/en/)
* [Katalon Docs: Home](https://docs.katalon.com/katalon-studio/docs/index.html)

Er is verder weinig aan te merken op de kwaliteit en beschikbaarheid van alle documentatie. Maar het is natuurlijk wel een belangrijk onderdeel om naar gekeken te hebben. Slechte documentatie kan de workflow van een developer op een slechte manier beïnvloeden.

# Conclusie & advies aan de klant

Afhankelijk van de technische capabiliteiten van de developers die de geautomatiseerde tests gaan schrijven zal een advies gegeven worden.

Voor developers met veel codeer ervaring raden we Selenium aan. De extra functionaliteiten die Katalon aanbied zijn over het algemeen gebruiksgemakken, daarom raden we Katalon aan voor gebruikers met minder ervaring met development van code.

Dus gebaseerd op wat developers fijn vinden werken en waar ze zich comfortabel mee voelen kan keuze gemaakt worden tussen katalon en selenium. Beide hebben vergelijkbare functionaliteiten, ze zijn alleen iets anders in gebruik.