Testplan – VI Labels



Schrijver: Oscar Veldman  
Bedrijf: VI Company  
Datum: 11 oktober 2016  
Updated: 10-1-2017  
Versie: 5.0

Table of Contents

[Small description 3](#_Toc471827446)

[Purpose 3](#_Toc471827447)

[Software 3](#_Toc471827448)

[responsibility 3](#_Toc471827449)

[Scoop 4](#_Toc471827450)

[Risks 4](#_Toc471827451)

[Tools 4](#_Toc471827452)

[Expectation 4](#_Toc471827453)

## Small description

Er komt een product dat het vertalen van items vergemakkelijkt. Je moet snel kunnen toevoegen, wijzigen en verwijderen. Ook wordt alles op alfabetische volgorde gezet in de JSON-file. Daarnaast moet je snel de items kunnen vinden. Verder worden de oude projecten geconverteerd naar de nieuwe versie. Dit worden de belangrijke functies van het product.

## Purpose

We willen alles gaan testen, omdat we zo snel mogelijk willen zien of er fouten aanwezig zijn in een programma. Daarnaast als er nog in de toekomst aan ons programma gewerkt wordt, dat niet het hele product stuk gaat per ongeluk. Hiermee schilt het ons een hoop werk in de toekomst verwachten wij. De tijd die we over hebben, die kunnen we dan weer gebruiken voor andere bezigheden. Het doel van het document is dat we van tevoren hebben nagedacht hoe we het gaan aanpakken en dat je achteraf kan zien wat er gedaan is. Als laatste willen we ermee bereiken dat de gebruiker zo min mogelijk hinder heeft van bugs, zodat hij ook gewoon door kan blijven werken.

## Software

We willen de meest belangrijke functies getest hebben van ons product. Ten eerste moeten ze ge-unittest worden. Als de file system aangeroepen wordt, dan moet er een mock file system aangeroepen worden. Dit moet omdat we willen geen andere details mee gewogen worden in de tests. Voor de unittesten gaan we n-unit gebruiken. Dit framework kunnen we dan ook gebruiken voor het automatische testen. We gaan de grote UI-functies testen. Voor de rest gaan we opgevangen door user experience te testen met andere developers die hier op kantoor werken.

Voor het systeem testen gaan we White UI Test framework gebruiken om de userinterface te testen. Op deze manier kan je door te programmeren de UI testen. Verder willen we de performence testen door de tijd bij te houden die het programma nodig heeft om de JSON-files te laden. Deze moet in een bepaalde tijd zijn geladen, zodat de gebruiker zo snel mogelijk kan gebinnen. Er wordt een file gegenereerd met meer als 1000 items/labels. Zodat we zeker weten dat het snel blijft. Een normaal project blijft flink onder de 1000 items.

Voor het automatische testen gaan we Bamboo gebruiken. Dit gaan we gebruiken omdat VI Company het ook gebruikt. Dit staat dan in dezelfde lijst als alle andere automatische testen. Je kan gelijk zien of er een test is mislukt. Ook krijgt iedereen die aan het project meewerkt een e-mail dat de tests zijn gefaald. Hierin kan je precies welke test het niet gehaald hebben. Er worden hierbij ook integratie testen gedaan voor sommige klassen en functies. Elke keer als er een commit wordt gedaan, wordt er even getest op de server. Op de server worden de tests en het programma gebouwd en vervolgens getest. Alle testen worden altijd uitgevoerd. Zodat we weten waar de fouten kunnen zitten. Er zal voor de rest geen andere automatische tests worden gedraaid.

## responsibility

Iedereen is verantwoordelijk voor zijn eigen code, dus als er een test faalt. Dan moet diegene het gaan oplossen. Dat kan zijn dat zijn code niet meer werkt, maar het kan ook zijn dat de test niet geschikt is. Denk hierbij aan dat de stakeholders van idee zijn veranderd. Omdat we met zijn tweeën zijn, kunnen we dit heel makkelijk aanpakken, want we zitten naast elkaar. Daardoor kunnen we heel makkelijk bespreken.

Oscar Veldman: [oscar@vicompany.nl](mailto:oscar@vicompany.nl)  
Jefrrey Hoogland: [Jeffrey@vicompany.nl](mailto:Jeffrey@vicompany.nl)

## Scoop

De applicatie wordt getest op een Windows 10 pc. Deze scoop houden we aan, omdat iedereen hier op kantoor heeft een windows 10 pc. Ook wordt niet alles getest, omdat sommige functies daarvoor te klein zijn en zeker weet dat het werkt.

De scenario’s die we testen zijn vooral hoe het systeem met de gegevens omgaat. Eén voorbeeld is het converteren van een oud project naar een nieuw project. Een ander voorbeeld is hoe ons programma omgaat met het verwerken van de JSON-files. We gaan ook kijken hoe snel ons programma gaat laden.

## Risks

De verwachte risico’s liggen vooral in dat we moeten voorkomen dat de unit testen geen onjuiste positieve reactie teruggeeft. De unittesten moeten goed gecontroleerd worden door ons zelf. De ui wordt minder getest. Dat komt omdat Dus de kans dat daar een bug voorkomt is groter, dan bij de gegevens verwerken. De migratie van de data worden verwerkt in het programma. Deze hoeven niet apart getest te worden.

## Tools

* Bamboo
* N-Unit
* White UI Test framework
* Analysis Tool for WPF Applications
* Shouldly (Nuget for N-Unit)

## Expectation

We verwachten dat we een product hebben van hoge kwaliteit. Waarbij de gebruiker alles zo snel mogelijk kan gebruiken. Ook moeten we voorkomen dat 1 functie de andere gaat breken. Zodat niet de hele programma crasht.