Scriptie ingediend tot het behalen van de graad van
PROFESSIONELE BACHELOR IN DE ELEKTRONICA-ICT

# Developing a framework for Coded UI Testing on the Windows 10 UWP

Peter Van de Putte, Jasper Van Gestel

academiejaar 2015-2016

AP Hogeschool Antwerpen
Wetenschap &Techniek
Elektronica-ICT

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

# Table of Contents


| | |
|---|---|
| Abstract | 1.1 |
| Inhoudstabel | 1.2 |
| Dankwoord | 1.3 |
| Introductie | 1.4 |
| Analyse | 1.5 |
| Design | 1.6 |
| Implementatie | 1.7 |
| Research | 1.8 |
| Conclusies | 1.9 |
| Glossary | 1.10 |
| Bibliografie | 1.11 |
| Appendices | 1.12 |

# Coded UI Testing

**Bachelor Elektronica & ICT - Peter Van de Putte & Jasper Van Gestel**

# Promotors


- **Stage promotor:**
  - Naam: Patrick Van Houtven
  - E-mail: patrick.vanhoutven@ap.be
  - Gsm: 0474 66 57 15
- **Stage mentor:**
  - Naam: Mark Devos
  - E-mail: Mark.devos@calidos.be
  - Gsm: 0475 69 13 42
- **Stage mentor:**
  - Naam: Tom Wuyts
  - E-mail: Tom.wuyts@calidos.be
  - Gsm: -


# Abstract


De Bachelorproef bestaat erin om Coded UI testing uit te voeren op het *'Maät'* project van *Calidos BVBA*.

De set doelstellingen bestaat onder meer uit:

- Leren werken met de testing tools, UWP en het Windows 10 OS. Dit aan de hand van kleine projecten die zelf worden opgezet of al reeds bestaan.
- Een "Guideline voor de tester" afleveren aan het eind van de stage die alle belangrijke informatie bevat omtrent gebruik, opstellen en werking van tests. Deze zal initieel kort zijn, maar zal in de loop van de bachelorproef verder aangroeien.
- Het gehele proces zo automatisch mogelijk te maken (Test automation).
- Naar aanleiding van het automatiseren is er halverwege de thesis een extra doelstelling bijgekomen:
  - Het creëren van Result Management Tools om een duidelijk beeld weer te geven van de progressiestatus van de testing op de applicatie.

# Dankwoord

Het ultieme dankwoord is niet zomaar een hoopje tekst waarin we iedereen bedanken, maar eerder de documenten die achterblijven om de opvolging en uitbreiding van het opgestarte project te verwezenlijken.

Dit gezegd zijnde, alsnog een "dikke merci" aan het Calidos-team dat het mogelijk -en aangenaam- maakten tijdens de stageperiode om het project tot een goed "einde" te brengen. Expliciet Tom Wuyts, onze stage mentor, die ons altijd voorzag van zijn kijk op de zaak. Ook Patrick Van Houtven, onze stage promotor, en Artesis Plantijn Hogeschool horen in het bedank-rijtje thuis.

Tot slot willen we ook elkaar bedanken, aangezien we samen op 12 weken tijd het ganse project van "null" hebben opgezet en waarbij dus een hele hoop teamwork kwam kijken.

# 1 Introductie

## 1.1 Situering

Het onderwerp van onze thesis is 'Coded UI Testing'.

Dit onderwerp werd toegepast op het "Maät" project, een Windows 10 applicatie die momenteel nog in ontwikkeling is door "Calidos".

> De naam "Maät" komt van de Oud-Egyptische godin "Maät" of "Ma'at", waar ze staat voor onder meer kosmische orde, waarheid en stabiliteit.

### 1.1.1 Stagebedrijf

Calidos is een IT-bedrijf, gevestigd in Mechelen, dat zich specialiseert in software voor de healthcare sector. Een KMO met veel groeipotentieel en een duidelijke visie op hoe de software nuttig en tegelijk overzichtelijk informatie moet weergeven om het dagelijkse werk bij ziekenhuizen praktisch en efficiënt te laten verlopen.


Eerder creëerde Calidos al een oplossing genaamd 'Othello' dat een toepassing is voor de MZG-registratie van VG-MZG scores (MVG-II) en personeelsgegevens. Meer dan 8750 ziekenhuisbedden hebben een licentie voor deze toepassing.

Het product ‘Team n Time’ laat ziekenhuizen toe personeel uit de mobiele equipe flexibel in te zetten in functie van hoe druk het is op een afdeling.

Het project waar deze thesis op toegepast wordt is het nieuwste project van Calidos, genaamd 'Maät'. Het Maät project is een Windows 10 applicatie die bedoelt is om clinical trials te plannen, te organiseren en op te volgen in ziekenhuizen.

## 1.1.2 Doelen

De vooropgestelde doelen van dit stageproject zijn:

1. Opleveren van een "Testing Guideline".
2. Opleveren van een "Testing Checklist" als onderdeel van de "Testing Guideline".
3. Opleveren van een "Testing Log" als onderdeel van de "Testing Guideline".
4. Opleveren van Result Management Tools *(Extra toevoeging op het einde van de stageperiode)*

Initieel waren deze documenten korte beschrijvingen, maar gedurende het verblijf bij "Calidos" zijn deze drastisch aangegroeid tot een duidelijk geheel dat in detail beschrijft hoe testing op "Maät" moet worden of, tot op zekere hoogte, werd verwezenlijkt.

### 1.1.2.1 Deliverables: Testing Guideline

De testing guideline moet alle documentatie bevatten voor personen die het project moeten verderzetten. In essentie zullen dus volgende vragen moeten beantwoord worden in dit document:

- Hoe moet de applicatie getest worden?
- Hoe moet testing automation toegepast worden op de applicatie?
- Hoe ga je best te werk als je hieraan moet beginnen?

Er wordt dus stap voor stap toegelicht hoe dit gebeurt en wat er nodig is om dit te verwezenlijken / op te volgen.

### 1.1.2.2 Opleveren van een "Testing Checklist" als onderdeel van de "Testing Guideline"

In essentie zal dus volgende vraag moeten beantwoord worden in dit deel van het document:

- Welke criteria moeten getest zijn om zeker te zijn dat de control / pagina volledig is uitgetest?

Opstellen van een lijst criteria, die "afgevinkt" kan worden en zo duidelijk weergeeft in hoeverre de applicatie getest is. De lijst zal dan in een soort matrix worden voorgesteld met langs de ene as een opsomming van paradigma's en sub-paradigma's en langs de andere as een opsomming van pagina's en controls.

Documenteren waar er zich problemen voordoen en wanneer mogelijk ook waarom. Ook problemen die zijn tegengekomen en hoe deze ondertussen verholpen zijn werden gedocumenteerd. Visuele foutjes, dingen die lichtjes verschillen op verschillende pagina's

maar in wezen wel dezelfde control zijn,... Al deze dingen staan in dit document beschreven.

> In de laatste weken van de stageperiode is er beslist om Result Management Tools te creëren. Deze tool zou dan dit gehele proces automatiseren (zie 'Result Management Tools' sectie).

### 1.1.2.3 Opleveren van een "Testing Log" als onderdeel van de "Testing Guideline"

In essentie zal dus volgende vraag moeten beantwoord worden in dit deel van het document:

- Welke bugs en niet werkende elementen / foutjes bij controls zijn er aanwezig in de applicatie?

Bijhouden van bugs en problemen en hoe deze initieel opgelost of omzeilt zijn geweest. Ook is er steeds gedocumenteerd welke pogingen ondernomen zijn om het probleem op te lossen zonder succes. Dit omdat het dan makkelijker is om bepaalde redeneringen uit te sluiten wanneer men opnieuw probeert om het probleem aan te pakken.

### 1.1.2.4 Result Management Tools

Als "zij-project" werd er de laatste 5 weken gevraagd of het mogelijk was om een kleine applicatie te schrijven die het mogelijk maakt om, via de build-straat, deze matrix weer te geven in "real-time". Waar we mee willen zeggen dat de matrix het resultaat zal zijn van drie afzonderlijke bestanden.

- Een "Definition" document
  - Definities van paradigma's op de ene en pagina's / controls op de andere matrix-as in XML formaat
- Een "Result" document
  - Resultaat van de testen, die 's nachts op de build-straat uitgevoerd werden in XML formaat
- Een "Target" document
  - Automatisch gegenereerd document. Dit gebeurt via de matrix applicatie, waar men manueel kan invoeren welke testen uitgevoerd moeten worden door de gewenste cellen op "TO DO" in te stellen.

Als resultaat krijgt men een matrix die opgesteld wordt via het "Definition" document. De cellen worden dan eerst ingevuld met het "Target" document en vervolgens overschreven (wanneer data beschikbaar is) met de resultaten uit het "Result" document.

### 1.1.2.5 Basiskennis

Vooraleer de echte doelen kunnen worden voltooid is het belangrijk te leren werken en te begrijpen met welk onderwerp we bezig zijn. Aangezien dit de eerste keer was dat we met testing in contact kwamen. De eerste fasen in onze thesis zijn dus:

- Uitzoeken wat "Coded UI Testing" nu eigenlijk is.
- Uitzoeken wat "Testing Automation" nu eigenlijk is.
- Leren werken met de Coded UI tools die voorzien zijn door Microsoft Visual Studio 2015.
    - De Coded UI Test Builder (Visual Studio 2015)
- Leren werken met het nieuwe OS van Microsoft, Windows 10.
    - Windows 10 (Microsoft)
- Leren werken met een UWP applicatie
    - UWP / XAML (Visual Studio 2015)

Wanneer de basiskennis opgenomen is, kan er gestart worden met het eigenlijke eindwerk dat ons gegeven is door Mark Devos.

## 1.1.3 Achtergrond

### 1.1.3.1 Voor- en nadelen van Coded UI Testing

Het gebruik van coded UI tests en test automation kan heel wat voordelen hebben voor een project. Om een idee te krijgen zijn hieronder de voornaamste opgelijst.

- Coded UI Tests zorgen voor stabiliteit in het project
    - Je kan aanpassingen doen aan je code zonder je teveel zorgen te maken over code die "breekt" of onverwachte bijeffecten. Bij elke aanpassing in code is het makkelijk (en bovendien ook goedkoop eens geschreven) om geschreven tests opnieuw te laten lopen. Zo ben je altijd zeker dat je project doet wat het moet doen.
- Coded UI Tests zorgen voor betrouwbaarheid in het project
    - Als er toch code "breekt" of functionaliteit veranderd, er altijd bekeken wordt via coded UI testing of er nog steeds gebeurt wat verwacht wordt. Het gekende "One step forward, two steps backward" wordt hier aangepakt door de resultaten van de testen te bekijken zodat je steeds op de hoogte bent van eventuele bugs of bijeffecten.
- Coded UI Tests zorgen voor schaalbaarheid in het project
    - Een enkele test laten lopen op het huidige project kan makkelijk handmatig. Maar waarschijnlijk zal je project wel meer dan één test nodig hebben om te garanderen dat het effectief doet wat het hoort te doen. Daarom is het makkelijk om testing automation te doen en deze tests automatisch te laten lopen zonder je zelf teveel zorgen te maken. Want, wanneer meerdere applicaties of projecten dezelfde bronnen gebruiken (een database of website bijvoorbeeld) los je dit makkelijk op

door deze tests (of delen ervan) over te nemen en ook hier te gebruiken.

Er zijn echter ook nadelen bij coded UI tests:

- Coded UI Tests hebben een redelijk hoge kostprijs
  - Om goede Coded UI Tests te schrijven is veel tijd nodig. Vooral wanneer het project veel veranderingen doorloopt tijdens de ontwikkeling van de applicatie of website op UI-vlak.
- Coded UI Tests vereisen een testplan
  - Een goed testplan opstellen kan een enorm verschil maken in tijd en kost. Daarom is het niet meteen een nadeel, maar men moet vooral onthouden dat het vooraf uitstippelen van de manier van testen heel wat problemen kan voorkomen.

# 1.2 Algemeen overzicht

## 1.2.1 Analyse

Om tot de bevindingen te komen in dit onderdeel was het noodzakelijk de testing tools en de applicatie te leren kennen en gebruiken. Wanneer men een basisbegrip heeft van deze onderwerpen kunnen we dieper ingaan op hoe testing effectief gebeurd.

## 1.2.2 Design

In dit deel wordt gesproken over de denkwijze die toegepast werd op alles rondom de thesis. Zo zal er aandacht geschonken worden aan onder andere de benamingen van tests, de hiërarchische opbouw van de applicatie "Maät" en de documentstructuur van het Coded UI Test project op de TFS.

## 1.2.3 Implementatie

In dit hoofdstuk worden de denkwijze en opbouw van de testing guideline beschreven. Zo zal er aandacht geschonken worden hoe de testing checklist opgebouwd is, telkens kort toegelicht per paradigma / sub-paradigma.

## 1.2.4 Research

Er was heel wat onderzoekwerk gebeurd die uiteindelijk niet geïmplementeerd is geweest omdat de uitkomst van het onderzoek niet positief was. In dit hoofdstuk worden deze onderzoeken besproken en bevindingen hieromtrent neergeschreven.

## 1.2.5 Conclusies

Hierin zal uitgebreid uitgelegd worden wat de conclusies zijn die men kan trekken uit deze bachelorproef. Alsook kort toelichten wat er in de toekomst gepland was als bepaalde zaken sneller en/of vlotter hadden vooruitgegaan.

### 1.2.6 Appendices

De appendices bestaan voornamelijk uit alle documenten die als "deliverable" worden gezien. We hebben het dus over de Testing Guideline, de Testing Checklist, de Testing Log, de bevindingen voor de ontwikkelaar en de tijdschatting om de applicatie volledig te testen.

## 1.3 Gebruikte tools en technologieën

Hieronder volgt een korte opsomming van de voornaamste technologieën die gebruikt zijn tijdens de stage.

### 1.3.1 Software

#### 1.3.1.1 Windows 10

Oorspronkelijk is het Maät project gestart met als doelplatform Windows 8, maar door de snelle upgrade van Microsoft naar het veel recentere Windows 10 is Calidos ook overgegaan om Windows 10 tot het doelplatform te maken.

#### 1.3.1.2 Visual Studio 2015

Visual Studio 2015 is een rijk, geïntegreerd ontwikkelingsplatform om applicaties te creëren voor Windows, Android en iOS. Maar ook webapplicaties en cloud services vallen onder deze noemer.

#### 1.3.1.3 Team Foundation Server (TFS) 2013

Team Foundation Server 2013 is een server die het praktisch maakt om code te delen binnen een bedrijf/groep. Het is te vergelijken met bijvoorbeeld GitHub. Het maakt het mogelijk om code te mergen naar de server vanop ieders eigen branch, alsook de historie bekijken van elke methode/project. De ideale uitbreiding op de Visual Studio IDE wanneer men in een team aan een groot project werkt.

#### 1.3.1.4 Maät

De applicatie "Maät" is, zoals al eerder vermeld, een Windows 10 UWP applicatie die nog steeds onder ontwikkeling is. De applicatie maakt het mogelijk om clinical trials te plannen, te organiseren en op te volgen in ziekenhuizen.

De applicatie bevat heel wat data die gestructureerd en overzichtelijk wordt weergegeven op zijn verschillende schermen. Er zijn Hub-schermen die het mogelijk maken om een overzicht van een bepaald deel informatie te krijgen zoals bijvoorbeeld alle clinical trials. En er zijn dan weer andere schermen die veel specifieker data weergeven zoals bijvoorbeeld een specifieke trial.

De applicatie werkt op alle Windows-10-draaiende apparaten en vereist een internetverbinding om de connecties met de database live te houden. Dit zodat alle data die aangepast wordt via een scherm, rechtstreeks aangepast kan worden in de database zelf.

#### 1.3.1.5 Xsd2Code++

Xsd2code++ is een Add-in voor Microsoft Visual Studio die code genereerd uit XSD bestanden. Het maakt als het ware objecten in Visual Studio die beschreven zijn in XML-vorm in de XSD schema's. Deze tool is gebruikt in de thesis bij het creëren van HTML in de Result Management Tools.

### 1.3.2 Tools

#### 1.3.2.1 Coded UI Test Builder *(Test Tool from Visual Studio 2015)*

De Coded UI Test Builder is een onderdeel van het testing framework dat voorzien is door Visual Studio. Het is meer bepaald een testing tool die de ontwikkelaar in staat stelt om controls te mappen via UIMap's. De UIMap is een partial klasse die voor de helft automatisch gegenereerd wordt. Dit deel zorgt voor de mapping en eventueel voor assertions (beweringen) die kunnen controleren of een bepaalde voorwaarde voldoet. Een soort van "if" statement, maar dan voor het testing framework. De andere helft van de partial klasse is voor manuele aanpassingen. Hier komen we later duidelijker op terug.

### 1.3.3 Technologieën

#### 1.3.3.1 Universal Windows Platform (UWP)

Het Maät project is een UWP applicatie opgestelt in C# en XAML


Met de komst van Windows 10 werd UWP geïntroduceerd, dat het Windows Runtime model verder evolueert en het zo naar een verenigde Windows 10 core brengt. Als onderdeel van de core, brengt UWP een gezamenlijk app platform dat beschikbaar is op alle apparaten die Windows 10 runnen. Het UWP voorziet een gegarandeerde core API laag over apparaten. Dit wil zeggen dat men een enkel applicatie pakket kan maken dat kan geïnstalleerd worden op een waaier van apparaten. Bovendien voorziet de Windows Store een verenigd distributie kanaal dat, met dit single app pakket, alle apparaten waarop de applicatie kan draaien bereikbaar wordt.

### 1.3.3.2 "C#"

De Thesis is grotendeels opgesteld in C#. Dit heet te maken met het feit dat het grootste deel van applicatie in deze taal is geschreven, wat de duidelijkheid van de code verbeterd. C# is een object georiënteerde programmeertaal van Microsoft en is gebaseerd op de C++ programmeertaal. De essentie is dat het de gemakkelijkheid van Visual Basic wil combineren met de kracht van C++. Ook .NET is een onderdeel dat heel gerelateerd is met de C# taal.

### 1.3.3.3 XML

XML, of languit Extensive Markup Language, wordt gebruikt om data te beschrijven. Het is een flexibele manier om informatie formats te creëren. Meestal wordt XML gebruikt als bestanden voor databases.

### 1.3.3.4 XSD

XSD, of languit XML Schema Definition, is een W3C aanbeveling van hoe men elementen moet beschrijven in XML. Het handige aan XSD is dat het mogelijk wordt om zo XML te kunnen omzetten in bruikbare objecten in andere talen.

# 2 Analyse

De eerste fase in deze thesis: Het uitzoeken, het begrijpen en het leren werken met Coded UI tests, de Coded UI Testing tools en de Maät applicatie.

## 2.1 Definities

Om te kunnen begrijpen waar de thesis om draait is het dus essentieel te weten wat het onderwerp juist is. Daarom zijn volgende termen belangrijk:

**Wat is een "Coded UI Test"?**

Een Coded UI Test stelt ontwikkelaars in staat om tests te creëren die user interaction (UI) kunnen nabootsen / simuleren op software applicaties. De tests hebben onder meer betrekkingen tot:

- Controleren of de applicatie correct opstart
- Controleren of de navigatie naar specifieke pagina's / modules correct verloopt
- Controleren of de ingevoerde data de correcte output weergeeft / verwerkt
- Controleren of de besturingselementen werken zoals men verwacht

**Wat is "Testing Automation"?**

Software testing automation of gewoon test automation is het gehele proces waarbij software tools voorgedefinieerde tests op een software applicatie uitvoeren vooraleer het betreffende product in productie gaat.

## 2.2 De Coded UI Test Builder en UI Mapping

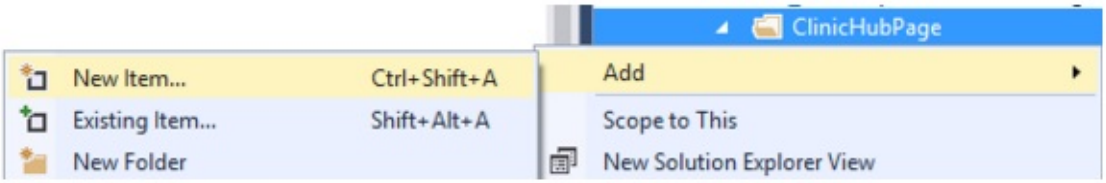
In deze sectie wordt beschreven hoe de coded UI Test Builder werkt en hoe UI Mapping gebeurt voor elk besturingselement. Er is heel wat tijd over gegaan vooraleer het duidelijk werd hoe deze testing tool werkte. Daarom zijn er ook regels en beschreven om de (toekomstige) code leesbaar en makkelijk aanpasbaar te maken en houden.

### 2.2.1 Een Coded UI Test klasse toevoegen aan het project

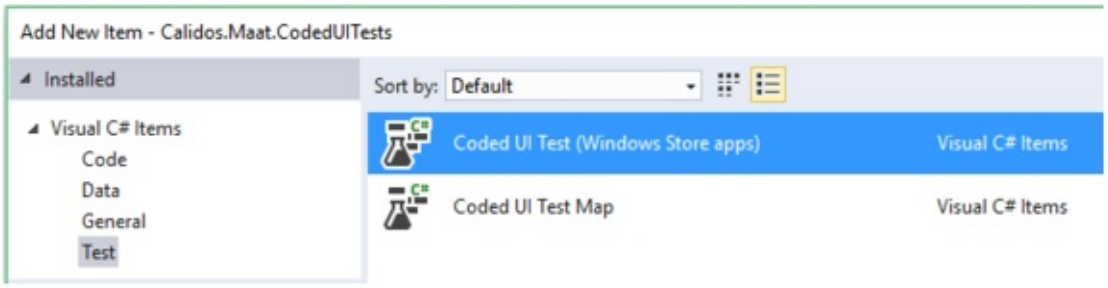
Om een Coded UI Test klasse toe te voegen aan het project zijn volgende stappen noodzakelijk:

- Rechtsklik op de map in het project waar de Coded UI Test gewenst wordt

- Selecteer "Add"
- Selecteer "New Item"




- Selecteer "Test"
- Selecteer "Coded UI Test (Windows Store apps)"




## 2.2.2 Een UI Map toevoegen aan het project

Om een UI map toe te voegen:

- Doe dezelfde stappen als bij "Coded UI Test klasse toevoegen"
- Selecteer "Coded UI Test Map" in plaats van "Coded UI Test (Windows Store apps)"

Wanneer men een Coded UI Test toevoegd aan het project is het belangrijk om steeds de juiste UI Map toe te voegen als variabele. Bovenaan de Coded UI Test klasse moet steeds een "using" statement toegevoegd worden voor de gecreerde UI Map. Als bijvoorbeeld de UI Map de naam “UIMap_ClinicHubPage” heeft moet er bovenaan staan:

```
using Calidos.Maat.CodedUITests.Screens.Clinic.ClinicHub.UIMap_ClinicHubPageClasses;
```

Onderaan de Coded UI Test klasse moet ook de UI Map property veranderd worden naar iets zoals volgende lijnen code:

```
public UIMap_ClinicHubPage UIMapClinicHub
{
  get
  {
    if (map == null)
    {
      map = new UIMap_ClinicHubPage();
    }
    return map;
  }
}
private UIMap_ClinicHubPage map;
```

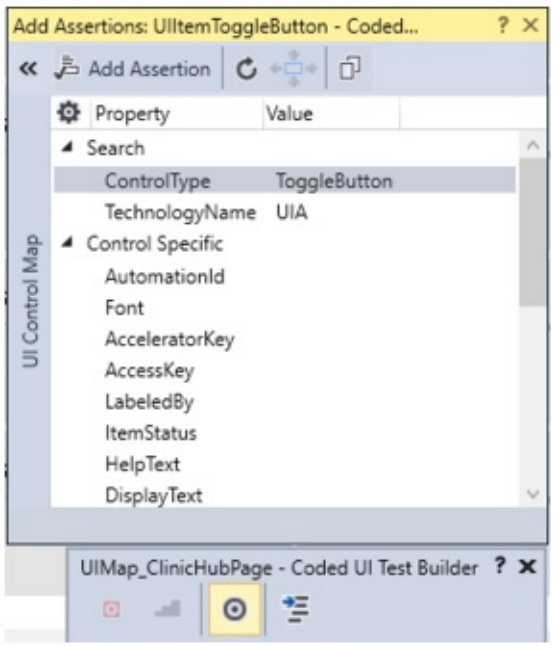
## 2.2.3 Mappen van besturingselementen

Als dit gebeurt is, is het mogelijk om tests te beginnen schrijven. Om een test te schrijven moet men eerst besturingselementen toevoegen aan de UI Map, het zogenaamde "Mappen van besturingselementen". Om dit te doen:

- Rechtsklik op het ".uitest" bestand
- Selecteer “Edit with Coded UI Test Builder”

De Coded UI Test Builder zal nu opstarten. Men moet zich vooral geen zorgen maken wanneer Visual Studio geminimaliseerd wordt. De Builder maakt hiermee duidelijk dat je eventueel een applicatie kan opstarten waarbij men besturingselementen wil mappen.

Het mappen van een besturingselement naar de UI Map gebeurt door de cirkelvormige marker ("add Assertions") te slepen naar de besturingselement. Wanneer de marker losgelaten wordt, wordt deze besturingselement opgelicht met een blauwe rand en zal er een nieuw venster verschijnen (Zie afbeelding onder).




Op het nieuwe venster kan meer informatie teruggevonden worden in verband met de geselecteerde besturingselement (eigenschappen). Als men op de pijl klikt bovenaan links in dit venster, zal het venster uitbreiden met een hiërarchie waarin de besturingselement zichtbaar wordt.

> **Nota:** Omdat de Clinical Trials applicatie ‘Maät’ gecreëerd is als een Windows 10 Metro App, is de Coded UI technology (voornamelijk de Coded UI Test Builder) nog niet volledig aangepast om de hiërarchie van besturingselementen correct te detecteren.
>
> Om de hiërarchie van besturingselementen correct te laten detecteren is het noodzakelijk om elke besturingselement een unieke automatisatie ID (UID) te geven. Op het moment van de stageopdracht was deze bij veel besturingselementen niet aanwezig, wat een bijkomend probleem opleverde. Meer hierover later.

## 2.2.4 Manueel UID's toewijzen

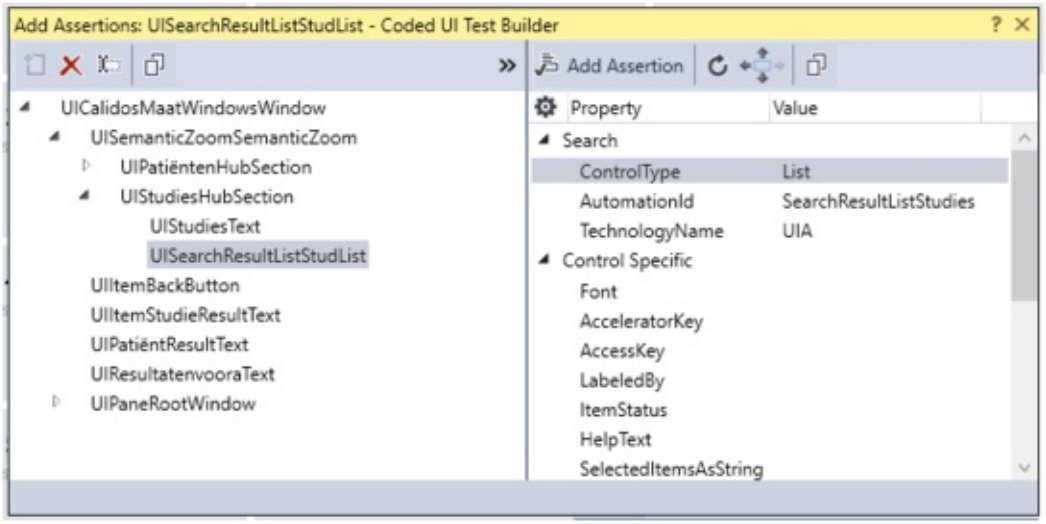
Om zeker te zijn dat besturingselementen juist gemapt zullen worden en makkelijk terug te vinden zijn in de toekomst door het testprogramma zelf, is het soms noodzakelijk dat men zelf de UID toewijst. Om dit te doen op een UWP applicatie:

- Open het XAML bestand waar de besturingselement zich bevindt
- Zoek naar de besturingselement in het XAML bestand
- Geef een AutomationId (UID) aan de besturingselement

In het voorbeeld hieronder werd een UID gegeven aan een list besturingselement. zodat de children van deze list later makkelijker gevonden kunnen worden.

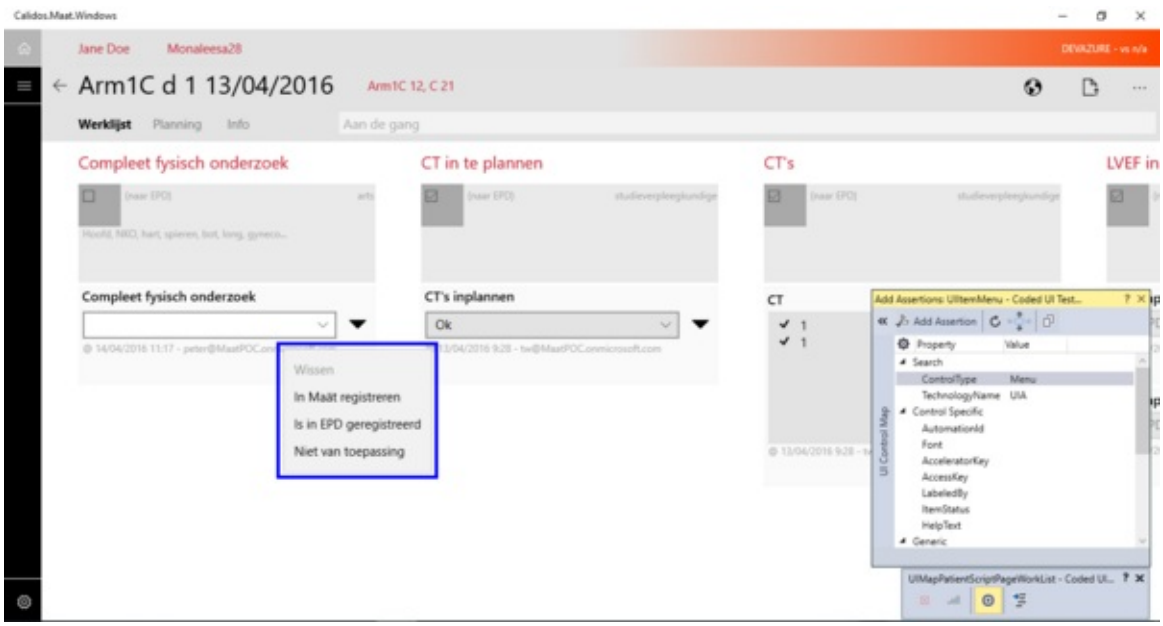
```
        <GridViewItem>Studies</GridViewItem>
        <GridViewItem>Patiënten</GridViewItem>
    </GridView>
</SemanticZoom.ZoomedOutView>
<SemanticZoom.ZoomedInView>
    <Hub Style="{StaticResource ContentHubStyle}">
        <interactivity:Interaction.Behaviors>
            <semantic:HubSemanticZoomProviderBehavior GroupsLink="{StaticResource semanticGroups
            <behaviors:ScrollToSectionBehavior x:Name="hubScroller" />
        </interactivity:Interaction.Behaviors>
        <HubSection Header="Studies"
                    Style="{StaticResource LeftMostHubSectionStyle}"
                    Visibility="{Binding SearchCompleted,
                                         Converter={StaticResource BoolToVisibilityConverter}}">
            <DataTemplate>
                <Grid Style="{StaticResource RootGridInHubSectionStyle}">
                    <Grid Style="{StaticResource ContentGridInRootGridStyle}">
                        <GridView x:Name="SearchResultListStudies"
                            IsItemClickEnabled="True"
                                  ItemTemplate="{StaticResource ITileDataTemplate}"
                                  ItemsSource="{Binding TrialResults}"
                                  SelectionMode="None"
                                  Style="{StaticResource GridViewBaseStyle}">
```

Wanneer de UID toegewezen is en men opnieuw de besturingselement selecteert met de marker zal de toegewezen naam zichtbaar zijn tussen de eigenschappen van de besturingselement.




## 2.2.5 Pop-up besturingselementen toevoegen aan de UI Map

Soms moeten er pop-up besturingselementen toegevoegd worden aan de UI Map. Dit kan niet zomaar door de marker te gebruiken. Een oplossing hiervoor is het gebruik van "Ctrl + i" wanneer men over de besturingselement zweeft die de pop-up voortbrengt. De Coded UI Test Builder zal hierdoor herkennen dat men de pop-up besturingselement wil selecteren.

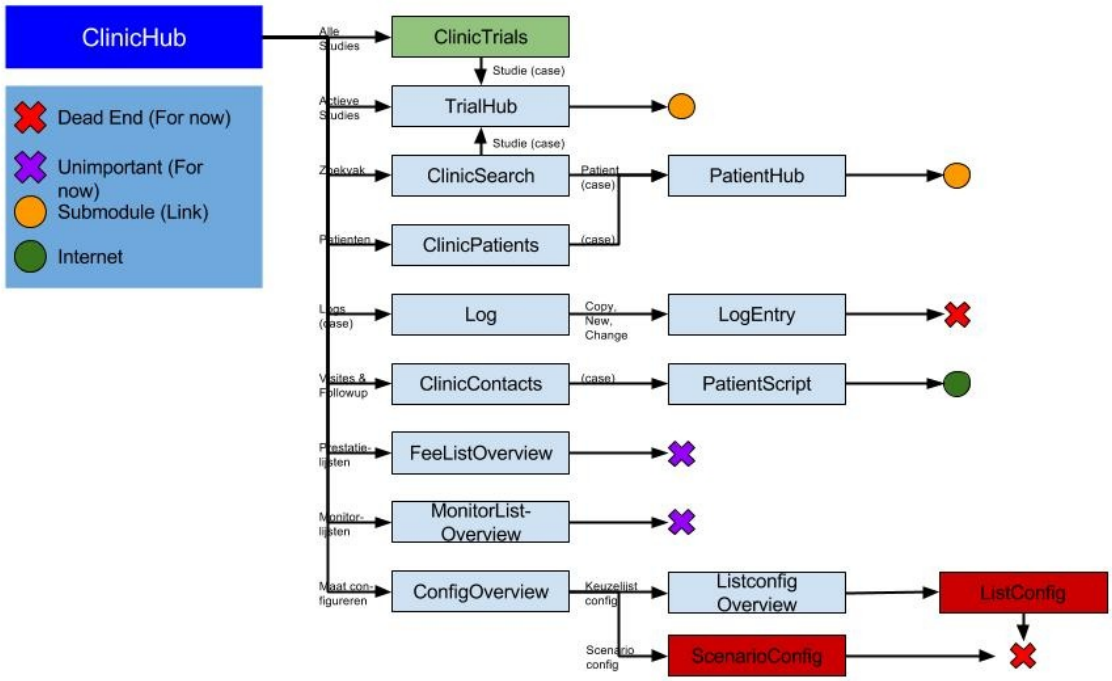



# 2.3 Maät ontdekken

## 2.3.1 Hiërarchisch navigatie ontwerp

In de eerste fase van de opdracht is er, zoals eerder vermeld, een hiërarchisch design opgesteld van alle navigatie die in de applicatie "Maät" mogelijk is. Dit gebeurde door alle navigatie uit te proberen en te documenteren naar welke pagina, of eventueel welke sub-pagina, een knop leidt. Zo kon er niet alleen een handig overzicht gecreëerd worden waarin we konden zien welke schermen het belangrijkste zijn, maar ook welke (voorlopig) minder belangrijk waren en gaf het een bijkomend voordeel, namelijk: de applicatie kon verkend worden. Wat dan weer handig was voor het verdere verloop van het project.

Het hiërarchisch design vertrekt vanuit de "ClinicHubPage". Dit is de hoofdpagina waarop men terecht komt als de applicatie gestart wordt (Na het inloggen). Van hieruit is een boomstructuur getekend naar alle pagina's waarnaar navigatie mogelijk is vanuit de "ClinicHubPage" (Zie afbeelding onder voor voorbeeld). Vervolgens is er voor elke hub-pagina een nieuw bestand gemaakt waarin dezelfde werkwijze gevolgd is als bij de "ClinicHubPage", namelijk al deze pagina's laten vertakken vanuit de respectievelijke pagina.




Bij elke tak is (op de pijl) geschreven welke knop of handeling ervoor zorgt dat we op die specifieke pagina terecht komen. Ook is er per pagina de belangrijkheid aangeduid in het testgebeuren. Sommige pagina's zijn namelijk nog niet af, of zijn zelfs nog in hun beginfase. Dit zijn dan pagina's die in het totale testproject minder prioriteit hebben ten opzichte van de pagina's die wel al af zijn of belangrijke informatie bevatten.

> De rode vakken stellen data-heavy pagina's voor die vrij uitgebreid zijn. Dit type pagina's is vrij belangrijk in de applicatie omdat deze data vaak andere pagina's kan beïnvloeden.

# 2.4 Planning

## 2.4.1 Testing + checklist

Om de applicatie Maät te testen, moeten verschillende stappen doorlopen worden.

**Fase 1**

Het leren werken met de coded-UI technologie zelf, er kan immers niet getest worden zonder kennis over de werking van deze technologie. Vervolgens is het de bedoeling om deze technologie toe te passen op enkele kleine projectjes, om zo vertrouwd te geraken met de code en technieken.

**Fase 2**

Uitzoeken hoe de tests voor Maät het beste kunnen geschreven worden. Er wordt dan rekening gehouden met hoe de applicatie is opgebouwd, eventuele moeilijkheden die zich voordoen, wat de belangrijkste dingen zijn die getest moeten worden. Dit gebeurt simultaan met het opstellen van de testing checklist, waarbij alle elementen die getest moeten worden onder elkaar geschreven worden in verschillende categorieën. Bij het opstellen van de checklist zullen al een aantal tests geschreven worden (voor elk paradigma een aantal tests) maar wordt er nog niet zoveel in de breedte getest.

**Fase 3**

Wanneer de checklist volledig af is, wordt de applicatie in de breedte getest. Dit wil zeggen dat alle elementen in de checklist voor elke pagina geanalyseerd worden (moet dit getest worden of niet) en nadien ook allemaal getest worden. Deze kunnen dan allemaal als getest gemarkeerd worden in de checklist.

## 2.4.2 Testresultaten

Het schrijven van de tests is één ding. Maar wanneer deze tests zijn uitgevoerd, is het uiteraard ook de bedoeling dat er resultaten weergegeven kunnen worden. Daarom is het noodzakelijk om een tool te ontwikkelen die de testresultaten kan verwerken en op een overzichtelijke en eenvoudige manier kan rapporteren aan de projectleider.

# 3 Design

## 3.1 Benaming van testmethoden

In dit onderdeel worden een aantal regels beschreven waaraan de namen van alle geschreven testmethodes moeten voldoen. De visie op deze naamgeving is dat men aan de naam van een testmethode alle informatie over wat er getest wordt kan afleiden. Wanneer een verslag terugkomt van alle uitgevoerde tests en er bepaalde tests gefaald zijn, kan er meteen worden gezien welk paradigma in de applicatie niet werkt zoals het hoort door gewoon de naam van de testmethode te lezen. Waarom dit paradigma dan gefaald is wordt weggeschreven in een test-log, waar we later op terugkomen.

Na meerdere brainstorm sessies over de benaming van testmethoden zijn we tot het volgende resultaat gekomen:

$$Page name + Paradigm (+ Context) (+ Xaml Control Type Property) (+ Specific Info)$$

- **"Page name"**
  - *Verplichte parameter*
  - Duidt de pagina aan waarin de test zich bevind.
  - Bijvoorbeeld: "ClinicHub", "ClinicSearch", "TrialHub", "PatientScript",...
  - Deze parameter moet altijd in het begin van de testmethode geschreven worden, omdat we dan meteen weten op welke pagina er zich een bug bevind indien de test faalt.
- **"Paradigm"**
  - *Verplichte parameter*
  - Duidt op het paradigma dat getest wordt. Dit is de denkwijze/werkwijze die de applicatie volgt om tot een bepaald resultaat te komen.
  - Bijvoorbeeld: Navigate (navigeer naar een andere pagina), Check (controleer een bepaalde waarde),...
  - Deze parameter komt meteen na de pagina om een duidelijke afbakening te zien van wat er wel/niet werkt op welke pagina.
- **"Context"**
  - *Optionele parameter*
  - Duidt meer specifiek aan welke besturingselement of welk deel van de geteste pagina getest wordt.
  - Bijvoorbeeld: Wanneer je meerdere lijsten hebt, die elk een lijst van zoekresultaten bevatten, en waarbij elke lijst een ander soort zoekresultaat bevat, kan je "Context" interpreteren als het woord dat duidelijk maakt om welke lijst het gaat.

- **"Xaml Control Type Property"**
  - *Optionele parameter*
  - Duidt op het type besturingselement dat getest wordt.
  - Bijvoorbeeld: Wanneer je bij context al hebt aangeduid welk sub-deel van de applicatie getest wordt, kan het nog steeds zijn dat dit sub-deel meerdere besturingselementen of besturingselement-types bevat. Deze parameter duid dan duidelijk aan om welke besturingselement het gaat (indien er één specifiek besturingselement getest wordt).
- **"Specific Info"**
  - *Optionele parameter*
  - Duidt op de info die je bij sommige tests nodig hebt om volledig duidelijk te maken wat er exact getest wordt. Dit kan allerhande informatie zijn.
  - Bijvoorbeeld: Een bepaalde state waarin de pagina zich bevind voor die specifieke test.

> Het is noodzakelijk dat minstens één van de optionele parameters ook aanwezig is in de testmethodebenaming om verduidelijking te geven over de betreffende testmethode.

# 3.2 Bestandsstructuur

In dit onderdeel zijn een aantal regels opgesteld voor de onderverdeling van de verschillende testprojecten. De visie hier is dat niet alle tests van het gehele project in één grote klasse mogen geschreven worden (wat de onderhoudbaarheid nagenoeg onmogelijk maakt). Tegelijkertijd mocht er ook niet voor elke testmethode een nieuwe klasse aangemaakt worden, aangezien er dan teveel files zouden ontstaan.

Als gulden middenweg werd gekozen voor het aanmaken van een nieuwe klasse voor elk hoofdparadigma, per pagina. Voor elke pagina werd ook één UIMap-klasse gemaakt die gebruikt werd in alle testprojecten voor de betreffende pagina. Daarnaast werd beslist dat bij elke pagina die verschillende sub-tabs bevat, elke sub-tab als één volledige pagina beschouwd werd. Deze worden dus onderverdeeld alsof het aparte pagina's zijn.

Door bovenstaande regels in acht te nemen, bekwamen we volgende file structuur:

- **Screens**
  - Bestand dat alle UIMap's en Testklassen bevat die zich bevinden in sub-folders
  - **General UIMap**
    - UIMap die gebruikt wordt voor de base-klassen
  - **Partial base-klassen**
    - Base klassen die methoden definieert die overal terugkomen
  - **Groepering**

- Bijvoorbeeld: Clinic, Patient, ...
- **Base-klassen per groepering**
  - Bijvoorbeeld: BaseClassConfigPages
- **Pagina**
  - Bijvoorbeeld: ClinicContactsPage, ClinicHubPage,...
  - **Base-klassen per pagina**
    - Bijvoorbeeld: BaseClassWorkbookConfig
  - **UIMap**
    - UIMap van de betreffende pagina
  - **Klasse per hoofdparadigma**
    - **Staten**
      - Klasse die alle testen betreffende verschillende staten van een pagina bevat
    - **Navigatie**
      - Klasse die alle testen betreffende voorwaartse en achterwaartse navigatie naar pagina's bevat
    - **Funtionaliteit**
      - Klasse die alle testen betreffende functionaliteit / inter-functionaliteit van besturingselementen bevat
    - **Control state appearance**
      - Klasse die alle testen betreffende de staat van besturingselementen bevat
    - **Content**
      - Klasse die alle testen betreffende data in de database of applicatie bevat

# 4 Implementatie

## 4.1 Testing checklist

### 4.1.1 Overzicht

Dit is de kern van het project. Het was de bedoeling dat er een lijst werd opgesteld waarin alle paradigma's die getest moesten worden onder elkaar weergegeven werden. In de "testing guideline" staat voor elk element in deze lijst beschreven hoe dat specifieke paradigma moet getest worden. Daarnaast is er in een apart excel document voor elke pagina opgelijst welke specifieke paradigma's op die bepaalde pagina aanwezig zijn. Op die manier wordt er per pagina in een matrix bijgehouden of een bepaald paradigma volledig getest is, moet getest worden of niet van toepassing is.

> Naar het einde van de bachelorproef toe werd er een automatisatie van dit process in ontwikkeld, er werd gehoopt dat deze volledig af zou zijn tegen het einde van de stage. Uiteindelijk zijn de tools niet volledig afgeraakt. Al zijn ze wel functioneel. Meer informatie over deze tools vindt je terug in de "Gebruikte tools en technologieën" sectie van het "Introductie" hoofdstuk.

### 4.1.2 Werkwijze

Er werd gestart met de basispagina’s die de meest gebruikte paradigma's/besturingselementen bevatten (die zo goed als overal terugkomen).

Voor elke gevonden besturingselement/paradigma werd dan ge-analyseerd hoe deze zo effecient mogelijk getest kon worden. Deze methode werd dan gedocumenteerd zodat eventuele opvolgers deze altijd terug kunnen raadplegen wanneer nodig. Er werd ook gedocumenteerd welke besturingselementen en paradigma's reeds getest zijn geweest, tot alles getest is voor de specifieke pagina. Op het moment dat alle paradigma's op alle besturingselementen die deze paradigma's toepassen getest zijn, wordt een pagina als volledig getest verklaard.

Indien er op een volgende pagina weer een nieuwe besturingselement/paradigma tevoorschijn komt, zetten we deze bij onderaan onze checklist met gevonden besturingselementen/paradigma's. Het is dan weer opnieuw de bedoeling om uit te zoeken hoe deze getest kan worden en alle voorgaande pagina’s opnieuw af te gaan en te controleren of ook dit element aanwezig is op de pagina en te testen indien nodig, zodat de pagina weer als getest verklaard kan worden.

## 4.1.3 General checklist

Deze checklist bevat de algemene paradigma's die op elke pagina aanwezig zijn. Ze zijn onderverdeeld in een aantal subcategorieën, die later verder onderverdeeld zullen worden.

Op de general checklist staat voor elke pagina een kolom, met daaronder voor elk paradigma of dit volledig getest is voor deze pagina of niet. Men verklaard een pagina als volledig getest wanneer de pagina specifieke checklist, die verder besproken wordt, volledig is afgewerkt.

Onderaan de general checklist staat ook een legende met een letter en een kleur die de test-status voorstelt van een paradigma (bijvoorbeeld: Y van Yes = getest, D van Do = nog te doen, E van Error = probleem bij testen,...)

## 5.1.4 Paradigm checklist

Aangezien het de bedoeling is dat er paradigma's gaan gezocht worden, en dat er dan voor elk paradigma uitgezocht wordt hoe dit moet getest worden, en dit nadien gedocumenteerd moet worden, leek het handig om een aparte paradigma-checklist te maken. Hier staan alle paradigma's, onderverdeeld tot op het niveau van specifieke scenario's, die getest moeten worden. Vervolgens staat er een kolom naast deze lijst, met dezelfde kleurcode als in de legende. Deze geeft dus aan of dit paradigma ge-analyseerd is en dus bekend is hoe dit moet getest worden.

Wanneer er een nieuw paradigma bijkomt, zal deze dus altijd eerst op groen moeten komen in de paradigma-checklist, vooraleer dit getest kan worden op de rest van de pagina's. Uiteraard zal dit dan wel op één pagina al getest zijn, namelijk de pagina waarop dit paradigma gevonden is en ge-analyseerd is.

## 4.1.5 Example-page + Page checklists

Vervolgens is er de pagina-specifieke checklist. Hierin staan de algemene paradigma's van de general checklist verder onderverdeeld zoals bij de paradigma-checklist. Bovenaan wordt de pagina dan opgesplitst in al zijn aparte besturingselementen, die worden gebruikt om tests uit te voeren. Voor elk besturingselement is ook het parent-besturingselement gedocumenteerd, zodat het duidelijk is over welke besturingselement het gaat.

Op de example-page zijn enkele besturingselementen aanwezig die op elke pagina terugkomen, zoals de backbutton of de home-knop.

Wanneer een nieuw paradigma gevonden wordt, zal dit eerst in de paradigm checklist terecht komen. Vervolgens komt paradigma op de example-page en nadien op alle andere page-checklists. Wanneer er begonnen wordt met een nieuwe pagina te testen en dus een

page-checklist gemaakt wordt voor deze pagina, kan de example-page rechtstreeks gekopiërd en geplakt worden, en kunnen vervolgens alle pagina-specifieke besturingselementen toegevoegd worden.

Op de page checklist staat opnieuw aangeduid welk paradigma getest is en welk niet, volgens de kleurcode in de legende op de general checklist, met als verschil dat deze hier nog eens onderverdeeld worden per besturingselement. Er wordt dus voor elk besturingselement ge-analyseerd welk paradigma van toepassing is en dan wordt de status volgens de kleurcode aangeduid.

## 4.1.6 Full checklist (eerste versie)

In de guideline staat de volledig uitgeschreven versie van de checklist. Hierin staat voor elk puntje in de lijst beschreven wat het exact inhoud onder "what?". Dit is een korte beschrijving van wat er getest wordt en wat de elementen die in de test gebruikt worden horen te doen.

Vervolgens staat onder "how?" een stappenplan beschreven dat je moet volgen om die specifieke test uit te voeren.

In deze checklist hoort zo specifiek en zo duidelijk mogelijk te zijn, zodat het ook voor een eventuele opvolger helemaal duidelijk is wat exact de bedoeling is van deze test en wat de verschillende besturingselementen en elementen horen te doen wanneer ze gebruikt worden op de beschreven manier.

Merk op dat deze checklist later nogmaals veranderd is. Bij deze de uitleg over de eerste versie van de checklist (de uiteindelijke volgt nog):

## 4.1.6.1 Content

In deze sectie worden alle paradigma's beschreven die betrekking hebben tot de inhoud van besturingselementen en tekstvelden. Alle informatie die in de user-interface beschikbaar is die uit de database komt, of aangepast kan worden, komt onder het paradigma "content".

Er zijn 4 verschillende handelingen die uitgevoerd kunnen worden op info in de besturingselementen. Deze 4 staan beter bekend als de CRUD-acties. (Create, Read, Update, Delete)

#### A. Create

"Create" wil zeggen dat je nieuwe data gaat toevoegen aan de database. Een voorbeeld hiervan is het toevoegen van een nieuwe patiënt of een nieuwe studie. Je vult alle velden in die relevant zijn en klikt dan op "toevoegen", wat ervoor zorgt dat er in de database een

patiënt bijkomt die nog niet bestond.

Om dit paradigma te testen, moet eerst uitgezocht worden op welk deel van de applicatie het toevoegen van nieuwe data invloed heeft. Vervolgens kan data toegevoegd worden (door tekstveldjes in te vullen, staat beschreven in de guideline). tot slot moet gecontroleerd worden of de data veranderd is waar deze moest veranderen. Voor die controle is een "Read"-functie nodig, dus create en read zijn deels verweven met elkaar.

> Voorbeeld: Ik voeg een nieuwe studie toe. Dit doe ik door een overlay te openen waarin een aantal tekstveldjes staan en een knop "toevoegen". Ik vul deze veldjes in en klik op de knop. Vervolgens navigeer ik naar de pagina waar de lijst met alle studies staat en loop ik over deze lijst om te controleren dat de studie die ik net heb toegevoegd aanwezig is in de lijst. Wanneer al deze stappen succesvol zijn uitgevoerd, is mijn test geslaagd.

**B. Read**

"Read" wil zeggen dat je gaat controleren of bestaande data, die aanwezig zou moeten zijn in de user-interface, ook effectief aanwezig is. Om dit te testen moet uitgezocht worden hoe enerzijds tekstvelden en eventueel andere besturingselementen uitgelezen kunnen worden, en anderzijds hoe deze vergeleken kunnen worden met de effectieve data in de database.

Er zijn verschillende scenario's die onder "Read" vallen.

**Weergegeven data**

Buiten het zoek-algoritme, welke een speciaal geval van "read" is, bestaat een read-test eruit om te gaan controleren dat de data die weergegeven wordt in de user-interface correct is.

Een voorbeeld is dat er naar de TrialHubPage genavigeerd wordt. Hierin staat alle data die te maken heeft met een bepaalde studie. Deze pagina gedraagt zich qua functionaliteit altijd hetzelfde, maar de data die weergegeven wordt hangt af van de studie waarop geklikt is. De bedoeling van de read-test is dan om te gaan controleren of de data die zichtbaar is in de user-interface dezelfde is als de data die verwacht wordt na een bepaalde navigatie.

Dit kan enerzijds hardcoded gecontroleerd worden, door eerst manueel de navigatie uit te voeren en dan alle data die zichtbaar is in code te schrijven als assertions (assertions worden later uitgelegd). Anderzijds kunnen datadriven tests geschreven worden, waarbij de data die we gebruiken in de test afkomstig is van een database (wordt ook later uitgelegd).

**C. Update/delete**

Update wil zeggen dat je reeds bestaande data gaat aanpassen en deze aanpassingen opslaan.

Delete is het verwijderen van bestaande data in de database.

Update en delete vallen binnen deze applicatie onder dezelfde tab, aangezien het deleten van data in de UI een update naar de database stuurt met een indicatie dat deze data niet meer bestaat.

**D. Custom**

Onder deze tab worden alle speciale gevallen geplaatst die nog bij content horen. Onder deze speciale gevallen horen bijvoorbeeld: het zoek-algoritme, de partpickers (wordt zo meteen besproken),...

**Zoek-algoritme**

Dit het algoritme dat de zoekfunctie doet werken. Er zijn 3 verschillende testscenario's die hierop uitgevoerd moeten worden.

1. Er moet getest worden of alle mogelijke parameters waarop we kunnen zoeken zoekresultaten opleveren. Het kan bijvoorbeeld zijn dat het zoekalgoritme zo is ingesteld dat je kan zoeken op de studienaam, patientnaam,...

   Het soort parameters waarop gezocht kan worden moet manueel ge-analyseerd worden. Nadien worden hier tests voor geschreven.

2. Vervolgens moet gecontroleerd worden of alle zoekresultaten die weergegeven worden het zoekwoord bevatten dat ingegeven is. Hiervoor moet een speciaal soort read-functie geschreven worden, waar later meer over verteld wordt.
3. Als derde moeten gecontroleerd worden of alle objecten in de database, die voldoen aan de zoekterm, ook effectief worden weergegeven. Het is één ding dat alle zoekresultaten de zoekterm bevatten, maar het zou natuurlijk altijd kunnen dat een aantal zoekresultaten die in de database wel effectief bestaan, niet worden weergegeven. In dat geval zou de vorige test wel werken, maar zou er toch nog een fout in het zoek-algoritme zitten. Vandaar dat deze derde test noodzakelijk is om het zoek-algoritme volledig te testen.

**E. Part-pickers**

Part-pickers zijn speciale knoppen waarmee je een datum of een tijd kan instellen. Door op de knop te klikken, verschijnt er een popup-venster. In dit venster staan een aantal verschillende tabs. Elk van deze tabs bevat een aantal blokken met waarden gaande van bijvoorbeeld 0-30/maandag-vrijdag/januari-december/... Door over deze blokken te hoveren met de muis en te scrollen, verschuiven ze, waardoor je de waarde aanpast. Er is altijd 1 van tab geselecteerd die je kan aanpassen. Bij het hoveren met de muis wordt de tab waarover je hovert automatisch geselecteerd. Je kan echter ook met de pijltjestoetsen van

links naar rechts gaan om andere tabs te selecteren. Als je met de pijltjestoetsen van boven naar beneden gaat verschuif je de geselecteerde tab altijd met 2 waardes per keer. Met het scroll-wiel van de muis verplaats je de geselecteerde tab met 1 waarde per keer.

Onderaan het popup-venster staan 2 knoppen, een vinkje en een kruisje. Door op het vinkje te klikken accepteer je de datum/tijd die je net hebt ingesteld en zal deze verschijnen in de Part-picker waarmee je net gewerkt hebt. Door op het kruisje te klikken wordt de verandering geanuleerd en blijft de waarde van de part-picker staan zoals die voordien stond.

De aangepaste waarde accepteren kan ook door op enter te klikken.

Telkens je een waarde accepteert/weigert verdwijnt het popup-venster terug.

Het testen van deze partpickers is tot nu toe nog niet gelukt, aangezien er geen enkele manier gevonden is om in testcode de partpicker te kunnen zien.

## 4.1.6.2 Navigations

Het volgende grote paradigma zijn de navigaties binnen de applicatie. Onder navigaties vallen alle acties die ervoor zorgen dat de applicatie een ander scherm opendoet. Ook de zoekfunctie behoort tot navigaties, aangezien we hier ook naar een ander scherm gaan. Het verschil tussen de CRUD-tests voor de zoekfunctie en de navigatietests is echter dat bij de navigatietests niet gecontroleerd wordt welke zoekdata weergegeven wordt, maar enkel of er naar de zoekresultatenpagina genavigeerd wordt (ClinicSearch).

Om een navigatie te testen zijn er dus in grote lijnen 2 handelingen die we moeten uitvoeren. Enerzijds moete de actie uitvoerd worden die zorgt voor de navigatie (meestal klikken op een besturingselement). Anderzijds moeten gecontroleerd worden of de juiste pagina wordt geopend nadat deze actie is uitgevoerd.

Deze controle kan voor elke pagina anders zijn, maar de werkwijze is steeds dezelfde: er wordt een besturingselement of een set van besturingselementen die uniek zijn voor de desbetreffende pagina gezocht, en gecontroleerd of deze besturingselementen aanwezig zijn, of dat ze de juiste waarde bevatten (bijvoorbeeld: titels).

**A. Soorten navigations**

Paradigmagewijs zijn alle navigations natuurlijk hetzelfde. Maar in manier van testen zijn de navigaties verder onderverdeeld in sub-paradigma's, waarbij elk sub-paradigma een lichtjes andere manier van testen omvat.

**Variabele besturingselementen**

Hieronder valt het concept van een lijst, waarin zich allemaal verschillende gevallen bevinden van een bepaald opject. Dit kan bijvoorbeeld zijn: een lijst van studies, een lijst van patienten,... Het aantal items in de lijst staat nooit vast, aangezien het afhangt van hoeveel studies/patienten/... er zich in de database bevinden. Dit kan voortdurend wijzigen. Ook de tekst op deze besturingselementen hangt af van de data in de database.

Als je op één van de besturingselementen in deze lijst klikt, zal je altijd op dezelfde pagina terecht komen. Hoe deze pagina is ingevuld hangt echter af van de besturingselement waarop je geklikt hebt.

Om dit te testen moet er dus enerzijds voor gezorgd worden dat het besturingselement aanklikbaar is, en nadien moet gecontroleerd worden of de titel van de pagina naarwaar genavigeerd werd overeen stemt met de besturingselement waarop geklikt is.

**Vaste besturingselementen**

Vaste besturingselementen zijn besturingselementen die altijd op een pagina aanwezig zijn, ongeacht de data in de database. De navigatie is uniek voor elk van deze besturingselementen. Soms kan het wel zijn dat meerdere vaste besturingselementen naar dezelfde pagina navigeren maar ze zorgen dan elk apart voor een andere state van de desbetreffende pagina. Ze openen bijvoorbeeld allemaal een aparte tab van dezelfde pagina of zorgen ervoor dat de pagina anders ge-ordend is.

Het aantal vaste besturingselementen op een pagina is altijd dezelfde, en deze staan ook altijd op dezelfde plaats gepositioneerd, enkel kan het zijn dat de tekst in deze besturingselementen varieert op basis van de data die zich in de database bevind.

Om deze besturingselementen te testen moet ook geklikt worden op de besturingselement, maar de concrete klikfunctie voor deze testmethode zal lichtjes verschillen van de variabele besturingselementen, aangezien de manier om toegang te krijgen tot het besturingselement anders zal zijn. De controle of de navigatie juist gebeurd is is opnieuw een controle op de titel van de pagina waarnaar genavigeerd werd, en eventueel een controle op de state van deze pagina (bijvoorbeeld: staat de juiste tab open? Staan de elementen in de pagina juist ge-ordend? ...).

**Zoekfunctie**

De zoekfunctie vanuit de ClinicHubPage kan ook beschouwd worden als een navigatie. Als puur het navigatiegedeelte hiervan getest wordt, moet er geen rekening gehouden worden met het algoritme dat zorgt voor de correcte zoekresultaten, maar enkel met het feit dat er genavigeerd wordt naar de zoekresultatenpagina.

Opnieuw zal dit een klein verschil geven in het schrijven van code, aangezien er deze keer niet moet geklikt worden op een besturingselementen, maar er eerst een zoekwoord moet ingeven worden en nadien ge-enterd moet worden of geklikt moet worden op het vergrootglas naast het zoekvak.

De controle gebeurt opnieuw op de titel van de zoekresultatenpagina.

**Hyperlink-navigatie**

Op pagina's die data bevatten die te maken heeft met één bepaalde studie of één bepaalde patiënt (of eventueel nog andere objecten die in de toekomst zouden kunnen tevoorschijn komen), staat bovenaan steeds een hyperlink met de naam van dit object. Als hierop geklikt wordt, verschijnt de overzichtpagina van dat object (bijvoorbeeld: studie->TrialHub, patiënt->PatientHub, ...)

Het schrijven van navigatiecode zal hier opnieuw lichtjes verschillen omdat de toegankelijkheid van de hyperlink lichtjes verschilt van de vorige navigaties. De controle gebeurt opnieuw op de titel.

> Voor al deze verschillende navigaties wordt steeds onderzocht hoe het besturingselement gevonden kan worden in code, hoe een verwachte waarde gecreëerd kan worden aan de hand van welke gecontroleerd kan worden of de juiste navigatie uitgevoerd werd, hoe dan deze control gebruikt kan worden (meestal klikken, aangezien de verschillende functionaliteiten zoals tab-enter bij navigaties nog niet van belang zijn, deze komen later terug bij functionality) en hoe dan gecontroleerd kan worden dat deze verwachtte waarde aanwezig is na de navigatie. Na elk van deze tests wordt dan ook nog de omgekeerde test gedaan met de backbutton, opnieuw met een verwachtte waarde en een effectieve waarde. Dit zorgt ervoor dat alle mogelijke back-navigaties in de applicatie uiteindelijk getest zijn. Dit geheel wordt zoveel mogelijk in één grote functie per soort navigatie geschreven, zodat als nadien dit soort navigatie nog tevoorschijn komt, de geschreven functie gewoon éénmaal aangeroepen moet worden en er dus geen extra werk meer is.

## 4.1.6.3 States

De verschillende states van een pagina zijn de verschillende soorten toestanden waarin die pagina zich kan bevinden. Dit zijn:

- Semantic zoom
  - Zoomed in
  - Zoomed out
- Overlay
  - Overlay is open

    - Overlay is gesloten
- Filtering search
    - Verschillende orderingen
- Multiselect
    - Meerdere geselecteerde elementen

**A. Semantic zoom**

De semantic zoom is een parent-besturingselement, die de mogelijkheid bezit om zichzelf in en uit te zoomen. Meestal bevind er zich in de semantic zoom een hub, die onderverdeeld wordt in verschillende hubsecties. Dit zijn allemaal aparte blokken waarin zich een aantal besturingselementen bevinden. Bovenaan een hubsectie staat dan de titel van deze hubsectie. Het aantal hubsecties is niet van belang, en ook het aantal besturingselementen die in een hubsectie geimplementeerd worden is niet van belang. Dat zijn er zoveel of zo weinig als je zelf wil.

Wanneer naar een pagina met een semantic zoom genavigeerd wordt, staat deze automatisch ingezoomd. Alle hubsecties zijn dan volledig zichtbaar met hun titel en alle besturingselementen. Uitzoomen wordt gedaan door "Ctrl-", Ctrl & scrollen, klikken op de hubsectie-titels, PgUp/PgDn&Enter en Tab&Enter. Wanneer uitgezoomd wordt verdwijnen alle volledige hubsecties en komt er in de plaats een lijst met listitems tevoorschijn, waarin alle titels van de hubsecties weergegeven zijn. Zo kan er makkelijk genavigeerd worden naar een hubsectie die helemaal rechts op het scherm staat en dus nog niet zichtbaar was in de zoomed-in state (toen moest er naartoe gescrolld worden).

Het terug inzoomen kan op dezelfde manier, door te klikken op de listitems, "Ctrl+", Ctrl & scrollen, , PgUp/PgDn&Enter en Tab&Enter. Als er terug ingezoomd wordt op een bepaalde hubsectie zal deze links van het scherm getoond worden.

Wat er dus moet getest worden is dat deze semantic zoom altijd in- en uitzoomt wanneer de beschreven actie uitgevoerd wordt. Dit moet opnieuw één keer volledig manueel ge-analyseerd worden, en nadien op zo een manier beschreven worden in een functie dat deze functie voor alle pagina's die een semantic zoom bevatten bruikbaar is zonder moeite.

Wat echter ook moet getest worden bij de semantic zoom, is het feit dat de hubsectietitels die in de zoomed-in state weergegeven zijn, ook overeenkomen met de titels die te zien is in de lijst als we uitzoomen. Ook moet gecontroleerd worden of alle hubsecties aanwezig zijn in de zoomed-in en zoomed-out state en dat deze aantallen dus overeen komen.

Op sommige pagina's bevat de semantic zoom dan ook nog eens content die gebaseerd is op de content veranderingen in de gehele pagina. Zo is het bijvoorbeeld zo dat op de PatientScript page een checkbox in de semantic zoom staat, die aan uit uitgevinkt staat

afhankelijk van het feit dat bepaalde info in de pagina is ingevuld of nog leeg is. Er moet dus ook getest worden dat deze checkboxes bij de zoomed-in en de zoomed-out state overeen komen.

Ook deze inhoudelijke tests moeten eerst manueel uitgevoerd worden en daarna in een functie weggeschreven worden zodat deze later hergebruikt kan worden zonder teveel denkwerk.

**B. Overlay**

Een overlay is een extra stuk scherm dat bovenop een weergegeven scherm komt, wanneer er op een bepaalde knop geklikt wordt. De intentie van de overlay is dat bepaalde data toegevoegd kan worden (create) of aangepast kan worden (Update). Meestal bestaat de overlay uit een aantal inputveldjes en een uitvoer-knop. Door deze inputveldjes in te vullen en op de uitvoer-knop te klikken wordt de data die jij net hebt ingevuld toegevoegd of aangepast in de database.

Wat hier moet getest worden zijn verschillende dingen. Eerst en vooral moet getest worden of de overlay initieel niet zichtbaar is. Dan moet er gekeken worden dat de knop die de bedoeling heeft de overlay te openen dit ook effectief doet. Dan moeten er gecontroleerd worden of de functionaliteit op de overlay zelf werkt naar behoren. Dit kan bijvoorbeeld zijn dat de uitvoer-knop pas actief wordt als bepaalde veldjes zijn ingevuld. Als laatste moet er getest worden of de overlay ook terug sluit als er op de sluit-knop of de uitvoer-knop geklikt wordt. Als er op de uitvoerknop geklikt wordt moet er gecontroleerd worden of de data die ingevoerd is doorgevoerd wordt naar de applicatie.

## 4.1.6.4 Functionality

Onder functionaliteit valt: alles dat te maken heeft met hoe de applicatie werkt, hoe besturingselementen werken,...

**A. Speed**

Speed heeft alles te maken met de snelheid waarmee de pagina's geladen zijn. Er zijn twee speed paradigmas die getest worden.

- Reaction-speed
- Reactivity

**Reaction speed**

De "reaction-speed" of reactiesnelheid is de snelheid waarmee een pagina geladen wordt. Dit is een speciaal soort test, die niet slaagt of faalt, maar een bepaalde waarde moet teruggeven.

Om dit te testen moet dus eerst uitgezocht worden hoe je nagaat of een pagina geladen is of niet. Vervolgens moet er uitgezocht worden hoe dit kan getimed worden en tot slot hoe deze getimede tijd opgeslagen en gerapporteerd kan worden.

**Reactivity**

Reactiviteit is het kunnen gebruiken van besturingselementen vooraleer de pagina volledig geladen is.

Om dit te testen moet een manier gevonden worden om te controleren of de pagina al geladen is of nog niet, nadat het besturingselement gebruikt is. Als de vorige test succesvol uitgevoerd is, is bekend hoe gecontroleerd moet worden of een pagina geladen is of niet, dus zou dit voor deze test geen probleem mogen zijn.

**B. Scrolling**

De scroll-functie wordt bij heel veel verschillende soorten vensters gebruikt. Het wordt gebruikt binnen een hub, binnen een combobox, sommige dropdown of popup-menus,...

Om de scroll-functie te testen, moet een manier gevonden worden om te controleren of de besturingselementen op het scherm dat getest wordt verplaatsen of zijn verplaatst. Vervolgens moet dan een manier gezocht worden om de verschillende soorten scroll-functies uit te voeren. Deze zijn onderandere het muis-scroll-wiel, de scrollbar,...

**C. Control state verification**

Dit is het controleren of de visuele toestand van de besturingselementen is hoe deze hoort te zijn. Enkele toestanden kunnen zijn: enabled/disabled, de kleur, de helptext,...

Er zijn ook verschillende scenarios waarvoor deze toestanden moeten gecontroleerd worden. Je kunt klikken op een besturingselement, klikken en vasthouden, hoveren over het besturingselement, ... Ook heb je nog de initiele toestand waarin het besturingselement zich bevind.

De tests zullen dus bestaan uit 2 delen, in het eerste deel wordt het scenario gecreëerd dat getest gaat worden (initeel, hover,...) In het tweede deel wordt de toestand van het besturingselement gecontroleerd, zich bevindend in dit scenario.

**D. Control accessibility**

Als de toestand van een besturingselement getest wordt, wordt natuurlijk ook de toegankelijkheid getest. Dit wil zeggen dat er getest wordt of je het besturingselement kan selecteren door middel van de tab en pijltjes toetsen. Het werd duidelijk dat dit voor sommige besturingselementen mogelijk is maar voor andere niet. Indien het mogelijk is moet er uitgezocht worden hoe er gecontroleerd kan worden of er een stippelijn rondom het

besturingselement zichtbaar is wanneer deze geselecteerd is. Vervolgens moet er automatisch getabt worden of op de pijltjestoetsen gedrukt worden in code en gecontroleerd worden of op een gegeven moment dit besturingselement geselecteerd is.

Later wordt er ook gecontroleerd of het besturingselement gebruikt kan worden met enkele toetsen op het toetsenbord als deze geselecteerd is.

**E. Custom**

Onder custom valt alle functionaliteit die specifiek te maken heeft met de besturingselementen zelf, en voor elk type van besturingselement alle tests die enkel gelden voor dit soort besturingselement.

## 4.1.6.5 Config

Alle config-pagina's van Maät, zijn veruit de meest unieke pagina's in de applicatie. Dit zijn de pagina's met de meeste maar ook de meest complexe functionaliteit, die nergens anders in de applicatie te vinden is. Daarom is er beslist om voor deze pagina's een apart paradigma te maken, waarin al deze unieke gevallen beschreven worden en er automatische functies van gemaakt worden. Deze functies kunnen dan op alle config-pagina's toepast worden. Dit is mogelijk omdat veel besturingselementen en elementen op exact dezelfde plaats en in exact dezelfde hiërarchie voorkomen op al deze pagina's.

## 4.1.7 Full checklist (nieuw)

Na een tijdje te werken met deze checklist, begonnen meer en meer paradigmas in verschillende categorieën elkaar te overlappen. Daarom is de beslissing genomen om de checklist nogmaals aan te passen en deels uit te breiden, om zoveel mogelijk paradigma's apart te houden en zo dus een overzichtelijke checklist te creëren. Het resultaat was volgende nieuwe indeling:

## 4.1.7.1 Content

Content krijgt een nieuwe onderverdeling, waarbij alle normale CRUD-operaties onder 1 subcategorie "CRUD" worden geplaatst met dan de verdere onderverdeling in Create, Read, Update en Delete. Onder "normaal" verstaan we gewone content die weergegeven wordt door besturingselementen.

Alle "niet-normale" CRUD-operaties zijn interactieve functies zoals bijvoorbeeld de zoekfunctie. Deze worden onder de tab "Custom CRUD" geplaatst.

Resultaat:

- CRUD
  - Create
  - Read
  - Update
  - Delete
- Custom CRUD
  - General
  - Search algorithm

### 4.1.7.2 Navigations

Onder navigations waren er 2 subcategorieën, namelijk het navigeren zelf en het creëren van een bepaalde state van een pagina bij een navigatie. Het lijkt echter logisch dat ook de loadspeed (die voordien onder Functionality stond) ook bij navigations wordt geplaatst, aangezien dit iets is dat getest moet worden onmiddelijk na een navigatie.

Resultaat

- Navigate to page
- Navigate to page-state
- Loadspeed

### 4.1.7.3 States

Bij states zijn er een hele hoop aanpassingen gebeurd. Na het analyseren van de applicatie is duidelijk geworden dat alle mogelijke states buiten overlay-state enkel voor kunnen komen wanneer een pagina niet in overlay-state is. Daarom is dit paradigma nu onderverdeeld in twee grote hoofdcategorieën: Overlay-state en no-overlay-state. Alle andere mogelijke states komen dan onder no-overlay-state. Het resultaat ziet er als volgt uit:

Resultaat:

- No overlay state
  - Zoomed in state
    - Alle mogelijke zoom-acties vanuit een zoomed in state
  - Zoomed out state
    - Alle mogelijk zoom-acties vanuit een zoomed out state
  - Filtering search state (page ordering
  - Multiselect state
- Overlay state
  - Alle mogelijke overlay-acties in een overlay state

### 4.1.7.4 Control state appearance

Dit was een onderdeel van functionality in de oude checklist, maar het lijkt logischer om ook dit paradigma apart te plaatsen. Onder control state appearance vallen alle tests die de toestand van een besturingselement (zowel visueel als functioneel) gaan controleren. Dit kan bijvoorbeeld zijn: de kleur van besturingselementen, het feit dat deze ge-enabled zijn, de waarde die zich in het besturingselement bevind,... Deze toestand moet voor verschillende handelingen gecontroleerd worden. Volgende handelingen zijn alle handelingen die de state van een besturingselement kunnen veranderen:

- Initial (= de oorspronkelijke toestand, voor er een handeling uitgevoerd is)
- Hovered (= wanneer de muis over het besturingselement zweeft)
- Clicked (= wanneer er op het besturingselement geklikt is)
- Click & hold (= wanneer er op het besturingselement geklikt wordt maar de muis ingedrukt gehouden wordt)
- Filled in (= wanneer er data in het besturingselement wordt geplaatst door de gebruiker)

### 4.1.7.5 Control functionality

Hieronder vallen alle andere functionality tests van de oude checklist, maar deze zijn anders geördend en meer veralgemeend (niet meer specifiek per type besturingselement, maar gewoon algemene tests) aangezien de volledige analyse voor elk type besturingselement moet gebeuren.

De nieuwe indeling is als volgt:

- General: Hieronder vallen alle algemene functionaliteiten, namelijk het selecteren van een besturingselement door click/tab/pijltjestoetsen, en het gebruiken van een besturingselement met click/enter/spatie.
- Execution functionality: Hieronder zijn alle functionaliteiten geplaatst die mogelijk zijn met besturingselementen in de applicatie. Dit kan gaan van het doen verschijnen van andere besturingselementen tot het scrollen doorheen de childs van een besturingselement.

### 4.1.7.6 Custom functionality

Deze categorie was oorspronkelijk bedoeld voor de config-pagina's, maar de naam is veralgemeend aangezien er op andere pagina's ook nog speciale functionaliteit zou kunnen bestaan. Deze categorie is echter nog niet voldoende geanalyseerd.

## 4.2 Testing Maät

Tijdens het schrijven van tests zijn er verschillende moeilijkheden en dingen die het testen moeilijker maken aan het licht gekomen. Daarom bevat de guideline ook een sectie waarin voor elke pagina het testproces beschreven wordt. Hierin wordt dus beschreven hoe er voor de specifieke pagina in kwestie te werk gegaan wordt om alle tests te analyseren en hoe bepaalde tests moeten geschreven worden om deze zo autonoom mogelijk te maken. Ook als duidelijk wordt dat bepaalde tests niet kunnen geschreven worden door gebrek aan ondersteuning of bijvoorbeeld slechte UIMapping, wordt dit in deze sectie uitgelegd. Zo kan de eventuele opvolger van het project makkelijk de draad oppikken.

### 4.2.1 General workmethod

Hierin wordt het algemene plan van aanpak beschreven die voor elke pagina hetzelfde blijft. Zo staat er bijvoorbeeld beschreven hoe de baseclass voor elke pagina moet opgebouwd zijn en wat hier het nut van is (namelijk het aanroepen van variabelen zodat deze in alle testprojecten van deze pagina kunnen gebruikt worden, en eventueel het schrijven van private functies voor de desbetreffende pagina).

### 4.2.2 Page specific workmethod

Hier staat voor elke pagina apart beschreven hoe er te werk gegaan wordt om deze specifieke pagina te testen (of een specifieke groep van pagina's).

## 4.3 How To Test

In dit onderdeel word algemeen beschreven hoe een test opgebouwd wordt en hoe deze werkt. Hier gaat het dus niet meer om hoe de applicatie Maät getest moet worden, maar echt hoe het Coded UI framework van Visual Studio in elkaar zit.

### 4.3.1 Basics

In de basics wordt beschreven hoe een testproject aangemaakt wordt, welke parameters er in dit project moeten staan om dit te laten runnen, hoe je deze parameters moet aanroepen en dergelijke. Ook hoe besturingselementen toegevoegd worden aan een testproject wordt hier beschreven. Dit is dus eigelijk het eerste hoofdstuk dat de eventuele opvolger moet lezen om aan het project te kunnen beginnen. Zonder deze basis is het onmogelijk om de rest van de guideline te begrijpen.

### 4.3.2 Generating controls

Onder generating controls staat alles wat nodig is om besturingselementen in een test aan te roepen en waardes van dit besturingselement te verifiëren. Dit is uiteindelijk de essentie van alle tests, namelijk dat we bepaalde waardes gaan controleren op hun correctheid.

### 4.3.3 Commonly used variables and methods

Hier staan een aantal variabelen en methoden beschreven die vaak nodig zijn om bepaalde tests uit te voeren. Een voorbeeld hiervan is de "StopWatch"-variabele.

### 4.3.4 Commonly used controls

Dit hoofdstuk gaat al terug iets specifieker. Hier staan een aantal speciale besturingselementen beschreven die vaak terugkomen in de applicatie en vaak gebruikt worden in tests. Het kan bijvoorbeeld gaan om de "ProgressBar", die aangeeft wanneer een pagina volledig geladen is.

### 4.3.5 BaseClassCodedUI

In dit hoofdstuk staan wel iets specifiekere zaken beschreven rond de applicatie Maät. Om het effectieve testen van de applicatie zo efficient mogelijk te laten verlopen, zijn er functies beschreven in een algemene baseclass (namelijk BaseClassCodedUI) die bepaalde tests automatisch uitvoeren. Het is tijdens de analyse de taak van de tester om bepaalde paradigma's te gaan uitzoeken (bijvoorbeeld: navigatie naar een andere pagina). Wanneer geanalyseerd is hoe dit moet getest worden wordt dit in een functie geschreven in de BaseClassCodedUI, waardoor dit paradigma bij de volgende test in één of enkele lijnen kan getest worden in plaats van elke keer opnieuw een hele blok code te schrijven.

## 4.4 Result Management Tools

In dit hoofdstuk wordt beschreven hoe de result management tools tot stand zijn gekomen. Het draait hier voornamelijk over het feit dat er nooit echt een goed beeld kan worden gegeven van de status van testing. Deze tools zijn ontwikkeld om de ontwikkelaar en de klant te bewijzen dat tests wel degelijk uitgevoerd werden, en garanderen dat de resultaten onvervalst zijn.

Er zijn twee tools die dit mogelijk maken:

- TRX 2 XML Parser
- Testresult Parser

### 4.4.1 TRX 2 XML Parser

Wanneer Coded UI Tests uitgevoerd worden via MSTest of via Visual Studio zal er een .trx document gegenereerd worden. Dit document bevat alle resultaten en andere informatie omtrent de tests. Het .trx document is opgesteld in een XML-structuur. Omdat dit document teveel informatie bevat wordt er enkel de essentiele informatie uitgehaald en in een nieuw aangemaakt XML document gezet.

### 4.4.1.1 RegisterTest

RegisterTest is een methode, gedefineerd in de BaseTestClass, die altijd als eerste moet worden aangeroepen binnen een testmethode. Deze methode zorgt ervoor dat er nog meer noodzakelijke informatie in de XML terecht zal komen. Het neemt namenlijk de scherm en paradigma ID op in zijn methode en schrijft deze dan bij uitvoering van de test mee weg in het .trx document. Op deze manier is er een link tussen de test, het scherm en het paradigma waarbij een test-resultaat wordt gegeven.

### 4.4.1.2 Running tests

In dit onderdeeltje is kort uitgelegd hoe er manueel voor gezorgd werd dat het .trx bestand gegenereerd werd. De bedoeling is uiteraard in de toekomst dat dit mee in de build-straat komt en het zo automatisch gebeurd.

### 4.4.1.3 Running the TRX 2 XML tool

Omdat er manueel nog variabelen moeten meegegeven worden in het stadium waar de tool zich op het einde van de stage zich bevond, is er duidelijk beschreven hoe dit moet gebeuren.

### 4.4.1.4 Result XML

Deze XML zal het gegenereerde XML bestand zijn dat voortkomt uit het .trx bestand dat de tests zelf op hun beurt voortbrachten. Belangrijk is om te weten wat er in dit bestand aanwezig is. Het Result XML bestand bevat volgende syntax:

```
<Test TestId=”Test Name” CategoryId =”Category GUID” ObjectId=”Object GUID” ResultLabe
l=”Outcome”>
```

- TestId
  - De test naam (test methode naam) die gegeven was in de code van de test
- CategoryId
  - De ID van de category, geschreven als een GUID
- ObjectId

    - De ID van het object, geschreven als een GUID
- ResultLabel
    - Bevat één van de vier verschillende mogelijke statusen:
        - Passed (De test is geslaagd)
        - Failed (De test is gefaald)
        - Aborted (De test die uitgevoerd werd is manueel geanuleerd)
        - Not Executed (De test is niet uitgevoerd geweest)

## 4.4.2 Testresult Parser

Deze tool is de tool waar het allemaal om te doen is als men spreekt over het weergeven van de vooruitgang bij testing. Het neemt drie XML bestanden als input en genereerd hieruit een HTML bestand dat de stand van zake weergeeft.

## 4.4.2.1 HTML bestand: Result tabel

Om te begrijpen hoe de tool werkt is het noodzakelijk de output te begrijpen. In dit geval genereerd de tool een HTML bestand. Hierin wordt een tabel weergegeven die de progressie van het testen van de applicatie weergeeft. De tabel is opgebouwd uit twee assen die de schermen en paradigma's weerrgeven en heel wat cellen die de status van elke testcase weergeven. Een cel kent 5 staten:

- Passed (100% Completion of test case)
- Failed (< 100% Completion of test case)
- To Do (This test case does not have a test written for it)
- Not To Do (This test case does not need a test written for it)
- Unknown (This test case is not yet been analyzed)

## 4.4.2.2 Definition XML

De Definition XML is het document waar manueel de assen van de tabel worden gedefineerd. Hier worden de GUID's toegewezen aan elk scherm of besturingselement en elke paradigma. Het Definition XML bestand bevat volgende syntax:

```
<Category id="Category GUID" name="Paradigm" info="Description" level="">
<Object id="Object GUID" name="Screen / Control" info="Description" level="">
```

- CategoryId / ObjectId
    - Het ID, geschreven als een GUID
- Name
    - Paradigma of scherm/besturingselement naam

    - Moet uniek zijn
- Level
    - Het level object zal gecalculeerd worden en moet daarom dus niet gedefineerd worden aangezien het toch overschreven zal worden

## 4.4.2.3 Target XML

Het Target XML bestand defineerd een doel voor elke testcase. Het Target XML bestand bevat volgende syntax:

```
<Target TargetName="Description" CategoryId="Category GUID" ObjectId="Object GUID" Tar
getLabel="Label"/>
```

- TargetName
    - Descriptieve naam van de category of het object (in leesbare tekst)
    - Gedaan, enkel om het beter begrijpbaar te maken (GUID alleen is niet leesbaar genoeg)
- CategoryId
    - Het ID van de category, geschreven als een GUID
- ObjectId
    - Het ID van het object, geschreven als een GUID
- TargetLabel
    - Bevat één van vier mogelijke staten:
        - To Do (Deze testcase vereist een test)
        - Not To Do (Deze testcase vereist geen test)
        - Unknown (Deze testcase is nog niet geanalyseerd)
        - Done (Deze testcase is manueel toegekend als zijnde 100% compleet)

## 4.4.2.4 XSD Files

Achter elk XML bestand zit een XSD schema in deze tools. Deze zijn toegevoegd voor een zeer goede reden. Zoals besproken bij de gebruikte tools en software, werd in het project gebruik gemaakt van Xsd2Code++. Deze tool zorgt voor de omzetting van XML objecten naar objecten in Visual Studio. Deze objecten zijn van belang bij de opbouw van de matrix die de resultaten uiteindelijk zal weergeven in een HTML bestand.

## 4.4.2.5 Running the Testresult Parser tool

De tool werkt in verschillende fasen. Elk in aparte methoden gegoten om een maximaal overzicht te bewaren en een efficiente werking te garanderen.

**Fase 1: Genereren van matrix in het geheugen**

Bestaat uit drie methoden:

- ProcessCategories
  - Gebruikt Definition XML om categorieën te maken op de verticale as
- ProcessObjects
  - Gebruikt Definition XML om objecten te maken op de horizontale as
- CreateCells
  - Maakt cellen en houdt parents en children bij

**Fase 2: Matrix populeren**

Bestaat uit twee methoden:

- ProcessTargetData
  - Vult Target XML data in de matrix in aan de hand van de GUID's
- ProcessResultData
  - Vult Result XML data in de matrix in aan de hand van de GUID's

**Fase 3: HTML genereren**

Bestaat uit nog eens acht verschillende methoden

- ProduceHtml
  - CreateHtmlHead
    - Maken van metadata
  - CreateHtmlBody
    - Maken van de tabel, legende en lijst met gefaalde tests
  - CreateTestResultDiv
    - Maken van de tabel wrapper
  - CreateTestResultTableHead
    - Maken van de tabel header (Objecten/Schermen)
  - CreateTestResultTableBody
    - Maken van tabel body (Cellen / Categorieën)
  - CreateLegendDiv
    - Maken van legende wrapper
  - CreateFailedDiv
    - Maken van gefaalde tests wrapper

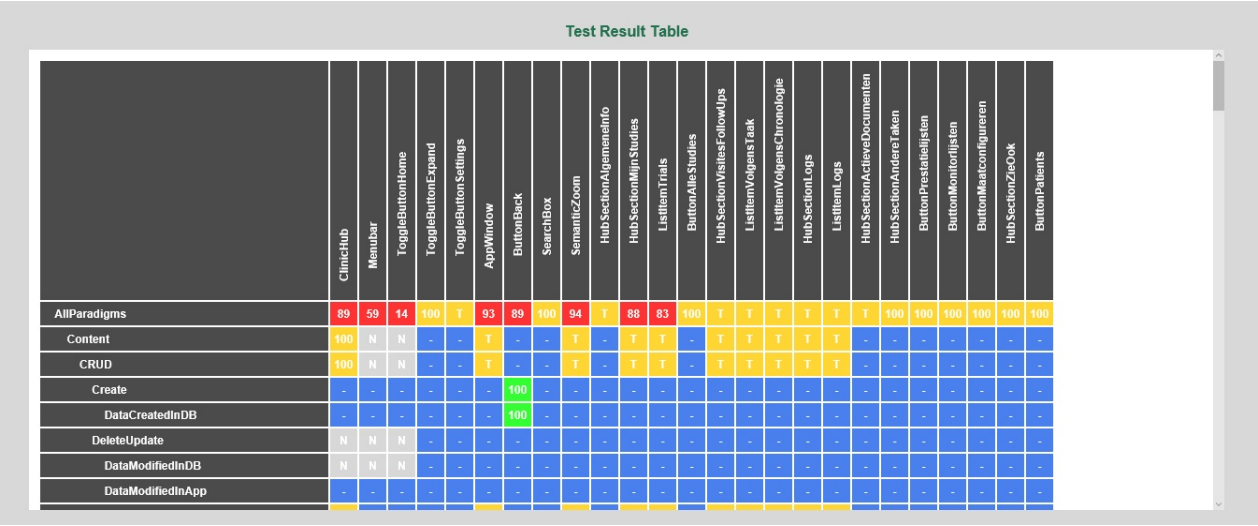
## 4.4.2.5 Extra functionaliteit

Er is ook veel tijd gekropen in het "bruikbaar" maken van de matrix. Een oneindig doorlopende reeks van schermen en besturingselementen langs de ene, en een hele hoop paradigmas op de andere as zou resulteren in miljoenen cellen. Deze moeten allemaal zichtbaar gemaakt **kunnen** worden. Maar het is niet handig om een overzicht te krijgen met miljoenen cellen als we maar kort willen kijken hoe de testprocedure ervoor staat. Dus, als oplossing, was het noodzakelijk dat de schermen en besturingselementen, alsook de paradigmas inklapbaar werden gemaakt zodat er een beter overzicht kon gecreëerd worden indien de gebruiker dit wenst.

In de toekomst was het ook gepland om de headers te laten zweven, zodat het ten alle tijden duidelijk was waar men zich bevond tijdens het scrollen. Heirvoor was echter niet genoeg tijd weggelegd.

## 4.4.2.6 Resultaat

Het resultaat is dan de HTML tabel met alle resultaten van de tests, zichtbaar in onderstaande afbeelding.



# 5 Research

## 5.1 Niet-geïmplementeerde research

### 5.1.1 Windows Store For Business (SFB)

#### 5.1.1.1 Opdracht

Tijdens de stage is er de vraag geweest van Calidos om een diepere kijk te nemen betreffende de Windows Store For Business met het oog op het distribueren van de applicatie. Enkele doelen werden opgelijst

- Hoe publiceer ik een applicatie op de Microsoft Store
- Hoe publiceer ik deze applicatie op de Windows Store For Business
- Hoe link ik een computer met de Store For Business
- Hoe link ik een computer met meerdere Strores Fro Business van meerdere bedrijven
- Hoe kan version upgrading gedaan worden, als de Store For Business gebruikt wordt
- Is er de mogelijkheid om Power Shell te gebruiken

Bijkomend na eerste fase in research:

- Hoe kan Windows Intune ons hierbij helpen

#### 5.1.1.2 Resultaten van research

Hieronder de conclusies die gelijst werden na diepgaand onderzoek naar de Store For Business.

> Het volledige document in verband met de research over de Store For Business is te vinden onder de "OverigeDocumenten/Documenten" folder

- Store for Business is in zeker zin een feature van de Windows Store
  - Implementeert dezelfde werkwijze als de Windows Store, met enkele voordelen specifiek voor een enkel bedrijf.
  - De voordelen van Store for Business houden onder andere in:
  - Alle apps van het bedrijf zijn zichtbaar voor alle/specifieke gebruikers binnen dat bedrijf
  - Flexibele distributie van deze apps op alle devices van het bedrijf
    - Private Store is een feature van de Windows Store for Business
- Windows Store for Business stelt in staat om apps in deze Private Store te zetten en

deze beschikbaar te maken voor een beperkt publiek (bedrijven)
- Enkel apps met online licenses kunnen in deze Private Store toegevoegd worden
- Het beschikbaar stellen van een app in de Private Store duurt ongeveer 12 uur
- Apps zonder certificaat kunnen in principe geïnstalleerd worden als de instellingen van de computer ingesteld staan als "modus voor ontwikkelaar"
    - Naar verluidt enkel beschikbaar als men ontwikkelaar programma's zoals VS op de computer heeft staan (de key zou deze modus activeren) > nog te controleren
    - "instellingen" typen in Windows search > Bijwerken & beveiligen > voor ontwikkelaars
    - Side-loading kan ook via deze weg ingesteld worden, aangezien er een "sideload modus" beschikbaar is onder deze tab
- Line-of-Business (LOB) lijkt een mogelijke oplossing voor het gestelde probleem
    - Via Mobile Device Management (MDM) zoals Windows Intune lijkt het mogelijk om updates gesynched te houden met alle devices
    - LOB kan via de Windows SFB
    - Windows Intune
        - Aparaatbeheer via cloud
        - Beschermen van bedrijfsdata
        - Devices registreren op bedrijfsnetwerk --> via portaal
        - Na registratie is mogelijk certificaten, VPN, WiFi in te stellen op deze devices
            - Toegang tot bedrijfsgegevens verschaffen
            - Eigen applicaties kunnen via de Intune App Wrapper ook gebruikt worden binnen dit systeem
            - Het pushen en beschikbaar stellen van apps, alsook hun updates is makkelijk beheerbaar.
                - Enkel de ingeschreven devices zijn in staat deze te instaleren
                - Ook meteen verwijderd wanneer werknemer of device niet meer in het bedrijf werkt/hoort
        - Kostprijs: Rond de 5 euro per maand per gebruiker (volgens Microsoft website, maar is onderhevig aan veranderingen.)

## 5.1.1.3 Bevindingen van Calidos

Er waren teveel factoren die verhinderde om de applicatie te distribueren en de applicatie tegelijk aan alle voorwaarden te laten voldoen die noodzakelijk waren. De Windows Store For Business nam 12 uur de tijd om een nieuwe versie te distribueren. Dit was te lang. Er werd dan gevraagd om het "Windows Intune" verhaal te bekijken. Deze deed ook niet volledig naar wens wat gezocht werd. Daarom is er beslist om andere mogelijkheden te bekijken.

## 5.1.2 Data Driven UI Tests

### 5.1.2.1 Wat zijn data driven UI tests

Alle variabele logica binnen de code van de UI tests zou altijd apart moeten worden gehouden, bijvoorbeeld in een database of datatabel. Deze data noemt men dan de test dataset of dataconfiguratie. Het voordeel hierbij is, dat men een grotere code coverage zal creëren op de betreffende UI test. Data driven tests zijn dus tests die op zich maar 1 enkele functie uittesten (gewoonlijk zijn dit lees en verificatietests) maar toegepast worden op een breed stuk code dankzij de dataset die ter beschikking gesteld wordt.

> In het onderzoek naar data driven tests is er gebruik gemaakt van CSV bestanden als datatabel (via Excel)

### 5.1.2.2 Hoe gebruik ik data driven UI tests

normale testmethode:

```
[TestMethod]
public void Method1()
{
    //testcode...
}
```

testmethode met datasource:

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV", "|DataDirectory|data.cs
v", "data#csv",
DataAccessMethod.Sequential), DeploymentItem("data.csv"), TestMethod]
public void ClickPatientSearchResults()
{
    // set searchproperty to value in datafile
    UIMap.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom.UIPatiëntenHubSection.
UIPatiëntenText.
    UIItemList.UICalidosMaatClientLogListItem1.SearchProperties[XamlControl.PropertyNa
mes.Instance] = TestContext.DataRow[0].ToString();

    // create control based on new searchproperty
    XamlControl Control = new XamlControl(UIMap.UICalidosMaatWindowsWindow.UISemanticZ
oomSemanticZoom
    .UIPatiëntenHubSection.UIPatiëntenText.UIItemList.UICalidosMaatClientLogListItem);

    //testcode...
}
```

De DataSource-functie heeft 4 variabelen nodig:

- Provider name: bij dit onderzoek was de provider CSV
  - Microsoft.VisualStudio.TestTools.DataSource.CSV
- Connectie string
- Tabel naam
- Data toegang methode

In deze testmethode is er een datafile gebruikt om de zoekeigenschap van een besturingselement die we willen uit testen te specificeren. Dit gebeurt in de CSV file zodat het mogelijk wordt om variabelen binnen de testcode te definiëren zonder effectief code te veranderen.

Het is mogelijk om kolom namen te vernoemen om de datakolom te specificeren op volgende wijze:

```
TestContext.DataRow["columnName"].ToString();
```

De testmethode zal de rest van de test herhalen voor elke rij data in de datafile.

## 5.1.2.3 Bevindingen

Er is vroeg gekeken naar data driven tests omdat het mogelijk was dat er van bij de start dan op een bepaalde manier gewerkt moest worden om de tests op te stellen. Er is heel kort de intentie geweest om zoveel mogelijk data driven te gaan doen. Hoewel dit absoluut aan de orde was geweest op lange termijn was het niet interessant genoeg om heel diep in het data-driven-test-wereldje te duiken en te concluderen dat alles getest zou zijn geweest van een welbepaald onderwerp, maar praktisch niets van de gehele applicatie. De documentatie over data driven tests is dus geen verloren moeite, maar het uitvoeren en in gebruik nemen van dit type tests zou eerder voor de toekomst zijn.

## 5.1.3 Test Automation (Integratie in Build-straat)

## 5.1.3.1 Probleemstelling

Het uitvoeren van Coded UI Tests werd lange tijd manueel gedaan op de computer die ook gebruikt werd om code te ontwikkelen. Wanneer een Coded UI Test uitgevoerd wordt is het belangrijk om alle randapparatuur beschikbaar te stellen aan de computer, zodat de tests naar behoren kunnen worden uitgevoerd. Als dit niet het geval is, zal de gebruiker de test verstoren en 95% van de tijd een verkeerd resultaat laten genereren door de testing tool. Zoals gezegd is men dus tijdelijk de computer kwijt en zit de ontwikkelaar/tester met een productief "gat" waar men niets meer kan doen dan wachten tot de alle tests uitgevoerd zijn geweest.

Omdat er een hele hoop tests geschreven waren en de tijdsduur van deze tests te laten uitvoeren steeds langer werd, werd beslist om te kijken of we deze in de build-straat konden krijgen. Dit hield in dat er virtual machines moesten aangemaakt worden en deze tests daar uit te laten voeren. Er waren dus enkele vragen die beantwoord moesten worden:

- Is het mogelijk om de tests op virtual machines uit te voeren
- Zijn er speciale vereisten om dit mogelijk te maken

### 5.1.3.2 Onderzoek naar implementatie in de build-straat

Er werd een virtual machine opgezet om dit te kunnen testen. Het zou te makkelijk geweest zijn als er geen probleem zou opduiken: Al doet men de uitvoer van de tests op een virtual machine, men heeft altijd een scherm, muis en toetsenbord nodig. Deze tests kunnen niet op de achtergrond worden uitgevoerd omdat deze dan niet meer overeenkomen met een "werkelijke" testomgeving.

Er was dus nood aan een beter concept om deze testing uit te voeren. Daarom werd gekeken naar testsettings, test agents en test controllers. Wanneer hiermee getest is geweest, doken er al snel (nog meer) problemen op. Het voornaamste en grootste probleem was, dat men een omgeving moest opstarten waar de applicatie kon gedeployed worden. Dit hield in dat men een virtual machine moest creëren waarop Windows 10 OS geïnstalleerd stond. Visual Studio 2015 moest ook aanwezig zijn om de test agents en test controllers op toe te kennen.

Omdat de testcontroller op de TFS moest aangemaakt worden (om zo meerdere testagents toe te wijzen) werd dit geprobeerd op de TFS 2013 van Calidos. Wanneer we spreken over 2013 wil dit zeggen dat in dit jaar nog helemaal geen spraken was van Windows 10. Met andere woorden. Een testcontroller aanmaken die Windows 10 als doel-OS heeft is hier dus onmogelijk.

### 5.1.3.3 Bevindingen

Vanwege het laatste probleem is ook dit onderzoek aan de kant geschoven. Al leverde dit wel een extra stimulans op voor Calidos om te overwegen om de TFS 2013 te upgraden naar TFS 2015 of TFS "in the Cloud".

Dit stukje kan ook teruggevonden worden in de Testing Guideline onder het hoofdstuk "Automation: Adding Coded UI Tests to the build street"

## 5.2 Geïmplementeerde research

## 5.2.1 Self-extracting zip

### 5.2.1.1 Probleemstelling

Door het update probleem dat zich nog steeds voordeed na de research naar de Windows Store For Business bleef het grootste probleem dat er niet genoeg vertrouwen is in de gebruiker om het update proces juist en makkelijk te laten verlopen. Calidos vroeg hierdoor om een oplossing te creëren in verband met het extracten van de geüpdate vesie.

Van Calidos uit gaven ze ook de hint mee om de "self-extracting zip" even te bekijken als oplossing voor dit probleem.

### 5.2.1.2 Oplossing

De self-extracting zip of SFX is een zip bestand dat zichzelf uitpakt en daarna zichzelf, via een extra script, kan installeren. Het voordeel hierbij is dat men zo veel mogelijk user interaction vermijd en garandeert dat alles correct geïnstalleerd wordt. De enige user interaction zal de bevestiging van uitpakken en installeren zijn.

Bij het uitzoeken naar welke library hiervoor het meest geschikt was zijn er 2 libraries de revue gepasseerd:

- SharpZipLib
  - GNU GPL Library voor .NET dat het mogelijk maakt om Zip's, GZip's, Tar's en BZip2's te creëren en uit te pakken
  - Volledig in C# geschreven.
- DotNetZip
  - Open software (Ms-PL) library voor .NET dat het mogelijk maakt om Zip's, BZip2's, CRC's en Zipstreams te creëren en uit te pakken.
  - Maakt gebruik van ionic Zip DLL.

Origineel is er gewerkt met de DotNetZip om Zip's aan te maken in het Maät project. Omdat er na een eerste onderzoek niet heel veel informatie beschikbaar was over een SFX binnen deze library, is geprobeerd met de SharpZipLib library te werken. Deze werkte vrijwel goed. Het nadeel was echter dat deze gebruik maakte van een extern opgestelde bitstream om de Zip te genereren. Deze bitstream zorgde voor problemen na enkele tests.

Na verder onderzoek is er beslist om toch met de DotNetZip library te werken. Er is verder gezocht naar bruikbare informatie omtrent het creëren van Zip's, die uiteindelijk ook werd gevonden. Na enkele tests met deze library werkte de code zoals gewenst. De code is

hieronder terug te vinden. Belangrijk om te weten is, dat de PostExtractCommandLine enkele keren herschreven is moeten worden omdat er enkele " / " of " " " misten of teveel geschreven waren.

## 5.2.1.3 Code: SFX creatie

```
private static void CreateZip(DirectoryInfo dirToZip, FileInfo zipFile, FileInfo exe)
{
    //Location where the SFX will extract to:
    string extractDir = @"%TEMP%/Maat";

    //We can't create an SFX with SharpZipLib (we do have added a 3rd party way to do
this, but this doesn't open the dir after unzipping.
    //DotNetZip nowadays allows for SFX's as well:
    using (ZipFile zip = new Ionic.Zip.ZipFile(zipFile.FullName))
    {
        zip.AddDirectory(dirToZip.FullName);
        zip.Comment = "Calidos Maät";
        var options = new SelfExtractorSaveOptions
        {
            Flavor = SelfExtractorFlavor.WinFormsApplication,
            DefaultExtractDirectory = extractDir,
            ExtractExistingFile = ExtractExistingFileAction.OverwriteSilently,
            RemoveUnpackedFilesAfterExecute = false,

            //Set to non-interactive, so the user doesn't have to do anything. We do d
efine the location where the files will be extracted to though.
            Quiet = true,

            //Launch the extracted path after unzipping:
            PostExtractCommandLine = "cmd /c start \"\" \"" + extractDir + "\"",

            SfxExeWindowTitle = "Extracting Maät Client...",
        };

        zip.SaveSelfExtractor(exe.FullName, options);
    }
}
```

# 6 Conclusies

Bij het begin van het project was de opdracht nog vrij vaag. Er werd gevraagd om de applicatie "Maät" te testen met behulp van de technologie "Coded User Interface Testing" (of afgekort Coded UI Testing). Het concept hiervan is simpel. Namelijk testen op zulke manier alsof een eindgebruiker met de applicatie aan het werken is. Er worden dus geen stukken code rechtstreeks getest maar alles gebeurt op een natuurgetrouwe manier via de UI.

De oorspronkelijke verwachting van het project was dat er 12 weken lang tests gingen geschreven worden om zo elk aspect van de applicatie te testen. Het stappenplan ging zijn om eerst te leren werken met de Coded UI-technologie, vervolgens dit uit te testen op kleine applicaties, en nadien op Maät te gaan werken. Ook was het oorspronkelijk de bedoeling om een rapport te schrijven voor Calidos met daarin alle opmerkingen over hoe de applicatie beter kan opgebouwd worden om het testen makkelijker te maken.

Na het leren werken met Coded-UI testing en enkele vergaderingen, werd het duidelijk dat er niet alleen getest moest worden, maar dat er ook een checklist moest opgesteld worden die alle te testen paradigma's bevat. Ook moest er een guideline opgesteld worden die de beschrijving bevat van hoe alles moet getest worden. In een log-document moesten dan alle tegengekomen fouten of moeilijkheden gedocumenteerd worden ter voorbereiding van het rapport voor Calidos.

Dit systeem ligt aan de basis van het schrijven van tests, maar na het schrijven van tests werd geconcludeerd dat dit alleen geen werkend systeem is. Er moet namelijk gerapporteerd kunnen worden wat de resultaten van de tests zijn, en dit op een overzichtelijke manier zodat de baas en de klant in één oogopslag kunnen zien hoever het ontwikkelen van de applicatie staat. Daarom is er nog een systeem bijgebouwd dat automatisch de uitgevoerde tests verwerkt en resultaten toont in een tabel op het scherm.

De algemene conclusie van het project is dus dat de opdracht in het begin vrij simpel leek, maar uiteindelijk toch veel meer bleek te omvatten dan oorspronkelijk gedacht.

# Glossary

## CSV

Comma Seperated Values

## GUID

Global Universal Identifiers

## Hub-pagina

Pagina's van waaruit we naar meerdere andere pagina's kunnen navigeren

## Maät

De applicatie van calidos die getest wordt.

## MZG

Minimale Ziekenhuis Gegevens

## Navigatie

Navigatie is wanneer men van de ene pagina naar de andere pagina gaat in een applicatie met behulp van een link, knop of actie. We spreken van connecties tussen verschillende pagina's wanneer ze naar elkaar kunnen navigeren.

## Paradigma

Een manier van doen, zienswijze

## SFX

Self-Extracting Zip

## UID

Unique (Automation) Identifier

## UWP

Universal Windows Platform

## VG

Verpleegkundige Gegevens

# 8 Bibliografie


- http://davidgiard.com/2012/07/24/GettingStartedWithCodedUITests.aspx
- http://searchsoftwarequality.techtarget.com/definition/automated-software-testing
- https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide
- https://msdn.microsoft.com/en-us/windows/uwp/index
- https://msdn.microsoft.com/en-us/windows/uwp/layout/design-and-ui-intro
- http://www.theregister.co.uk/2015/03/25/metro_meets_windows_10_can_microsoft_win_uap_preview/
- https://msdn.microsoft.com/nl-be/library/cc668205.aspx
- https://msdn.microsoft.com/en-us/library/dd286726.aspx
- https://msdn.microsoft.com/en-us/library/dd380782.aspx
- https://msdn.microsoft.com/en-us/library/dn305948.aspx
- https://msdn.microsoft.com/en-us/library/ff398056.aspx
- https://msdn.microsoft.com/en-us/library/ff398062.aspx
- https://msdn.microsoft.com/nl-be/library/ff400217.aspx
- https://msdn.microsoft.com/nl-be/library/ff400221.aspx
- https://msdn.microsoft.com/en-us/library/ff977233.aspx
- https://msdn.microsoft.com/en-us/library/gg269469.aspx
- https://msdn.microsoft.com/en-us/library/hh552522.aspx
- https://msdn.microsoft.com/en-us/library/windows/apps/hh454036.aspx
- https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide
- https://blogs.msdn.microsoft.com/gautamg/2010/01/05/3-introducing-sample-excel-extension/
- http://blogs.msdn.com/b/dpksinghal/archive/2011/09/28/how-to-test-deep-hierarchy-controls-using-coded-ui-test-in-wpf.aspx
- http://blogs.msdn.com/b/tapas_sahoos_blog/archive/2011/11/07/troubleshooting-record-and-playback-issues-in-coded-ui-test.aspx
- https://blogs.msdn.microsoft.com/tapas_sahoos_blog/2010/12/27/decoding-the-coded-ui-test-playback-failure-search-may-have-failed-at-controlx-as-it-may-have-virtualized-children/
- http://www.codeproject.com/Articles/172391/UIAutomation-Coded-UI-Tests-AutomationPeer-and-WPF
- http://stackoverflow.com/questions/10699795/vs-team-test-multiple-test-initialize-methods-in-test-class
- https://www.wikipedia.org/

- http://www.calidos.be/site/public/nl/Home/Home.aspx
- http://stackoverflow.com/questions/14304449/why-coded-ui-test-automation-is-important
- https://www.visualstudio.com/en-us/products/vs-2015-product-editions.aspx
- https://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx
- https://technet.microsoft.com/en-us/windows/store-for-business.aspx?f=255&MSPPError=-2147217396
- https://blogs.windows.com/buildingapps/2015/11/16/increase-your-apps-reach-with-windows-store-for-business/
- http://www.comparex-group.com/web/com/about/press/2016/setting-up-a-store-with-microsoft-store-for-business.htm
- https://technet.microsoft.com/nl-nl/library/mt592935%28v=vs.85%29.aspx
- https://channel9.msdn.com/Events/Windows/Developers-Guide-to-Windows-10-Version-1511/Windows-10-for-Business-Publishing-apps-to-the-Business-Store
- https://www.microsoft.com/nl-be/server-cloud/products/microsoft-intune/features.aspx
- https://www.youtube.com/watch?v=Hk8OmXWbG30
- https://technet.microsoft.com/en-us/library/hh441740.aspx
- https://icsharpcode.github.io/SharpZipLib/
- https://dotnetzip.codeplex.com/
- https://www.visualstudio.com/en-us/docs/test/lab-management/test-machines/install-configure-test-agents
- http://www.xsd2code.com/
- http://searchwindevelopment.techtarget.com/definition/C
- http://searchsoa.techtarget.com/definition/XSD
- http://searchsoa.techtarget.com/definition/XML

# 9 Appendices

Het *appendices hoofdstuk bevat volgende documenten:

**Testing Guideline**

De hoofd-deliverable van de thesis. Bevat alle informatie omtrent:

- Waarom testen
- Hoe testen
- Regels omtrent opstellen van tests
- Documentatie over ontwikkeling van tests

**Log**

Deliverable van de thesis. Bevat alle informatie omtrent:

- Problemen
- Bugs
- Oplossingen en work-arounds
- Dingen die geprobeerd zijn om problemen zonder succes op te lossen

**Testing Rapport**

Dit document bevat aanbevelingen over verbeteringen die aan de applicatie aangebracht zouden kunnen worden, kijkend vanuit het standpunt van de tester. Het zijn, als het ware, aanbevelingen die het makkelijker zouden maken voor testing doeleinden.

**Tijdsschatting**

Schatting van de noodzakelijke tijd (in dagen) om de volledige applicatie getest te krijgen.

**Windows SFB Research Notes**

Notities over het onderzoek naar de Windows Store For Business, Windows Intune en alles wat daarbij kwam kijken.

****Testing Checklist**

Het is een Excel bestand dat de eerste versie van de checklist bevat. Aangezien later de Result Management Tools dit systeem moesten overnemen speelt dit ook iets minder er een rol. Maar het geeft een beeld van waar de thesis op een bepaald moment heeft gestaan, tegenover waar de thesis is afgerond.

****TestResultTable**

Dit is het uiteindelijke eindproduct van de Result Management Tools. Een HTML document dat een bevat tabel die de status weergeeft in hoeverre de applicatie getest is.

*Alle documenten zijn te vinden onder "OverigeDocumenten/Documenten".

**Deze documenten zijn niet aanwezig in de scriptie omdat het format dit niet toelaat.