

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



---

# 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.

## Abstract

---

# 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. Explicet 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 groepotentieel 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 aangepast 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-staat, 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-staat 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 opgeliist.

- 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 bekijken 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 opgesteld 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

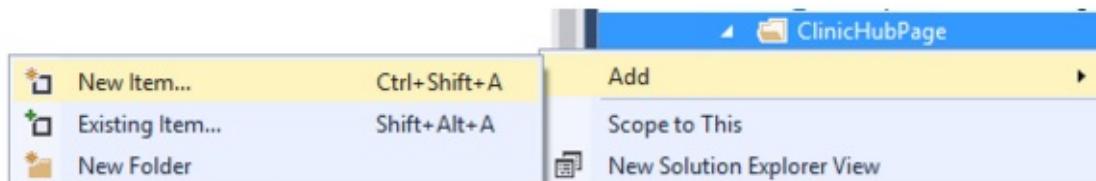
In deze sectie wordt beschreven hoe de coded UI Test Builder werkt en hoe UI Mapping gebeurt voor elk besturingselement. Er is heel wat tijd over gegaan vooraleer het duidelijk werd hoe deze testing tool werkte. Daarom zijn er ook regels en beschreven om de (toekomstige) code leesbaar en makkelijk aanpasbaar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

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

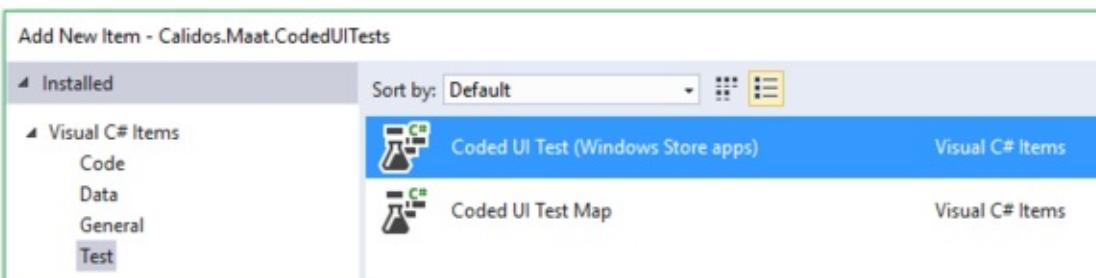
Om een Coded UI Test klasse toe te voegen aan het project zijn volgende stappen noodzakelijk:

- 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;
```

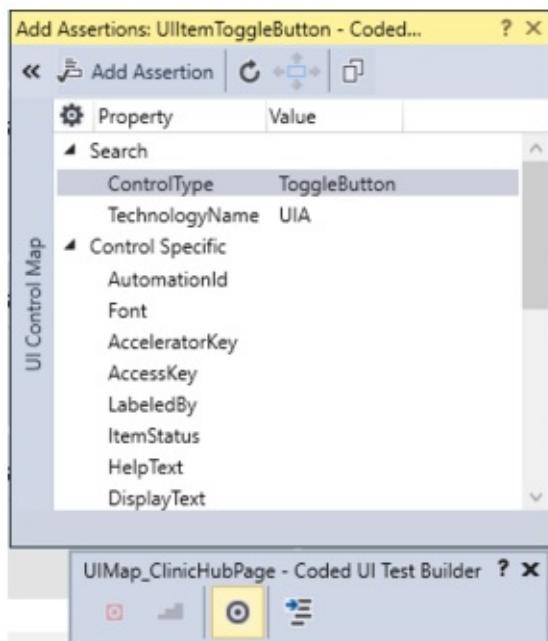
### 2.2.3 Mappen van besturingselementen

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

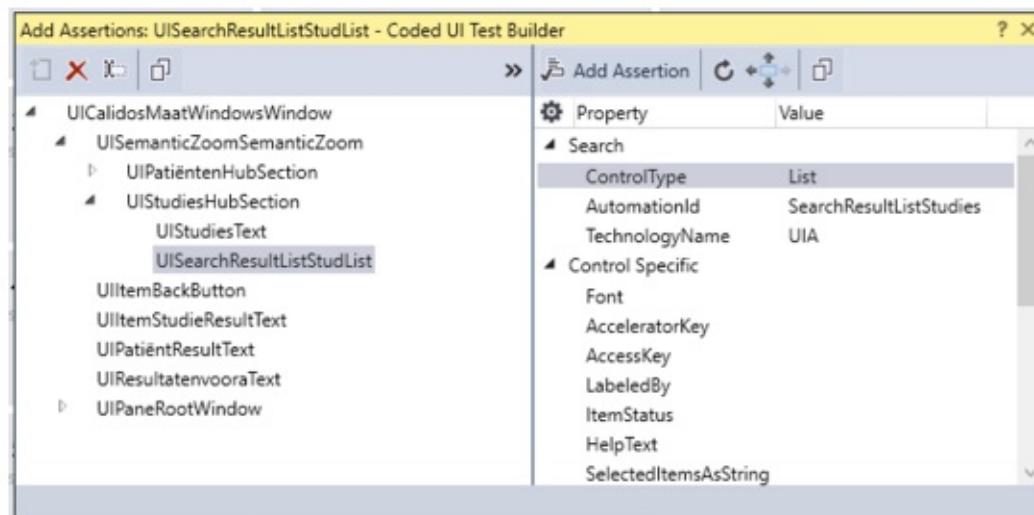
In het voorbeeld hieronder werd een UID gegeven aan een list besturingselement. zodat de children van deze list later makkelijker gevonden kunnen worden.

```

        <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}">
                    </Grid>
                </Grid>
            </DataTemplate>
        </HubSection>
    </Hub>
</SemanticZoom>

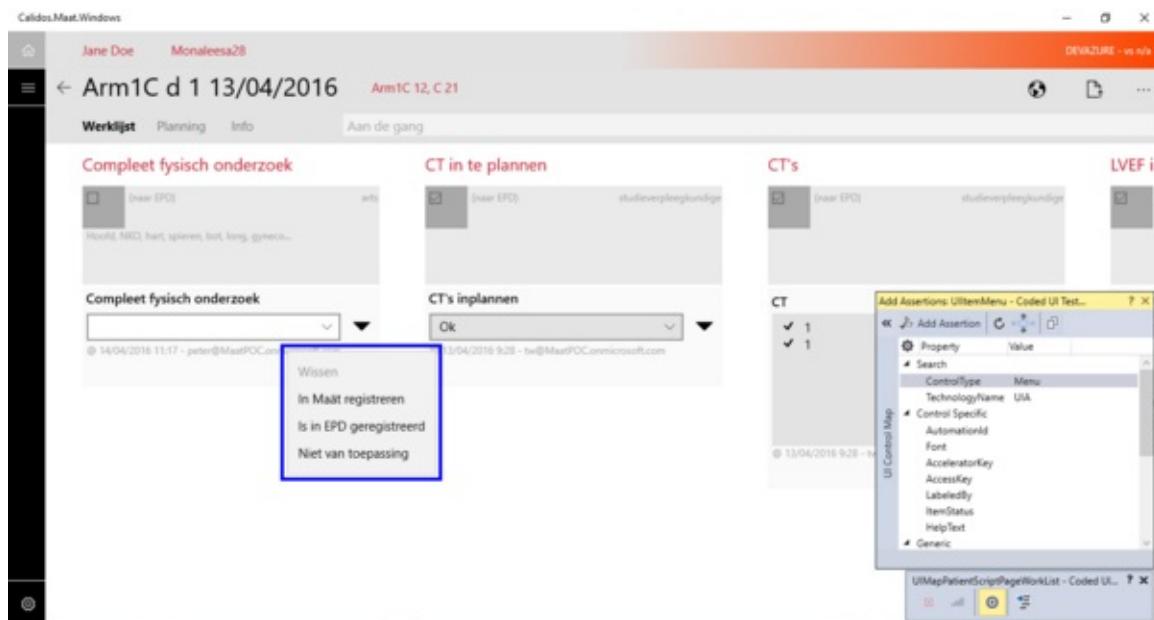
```

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.

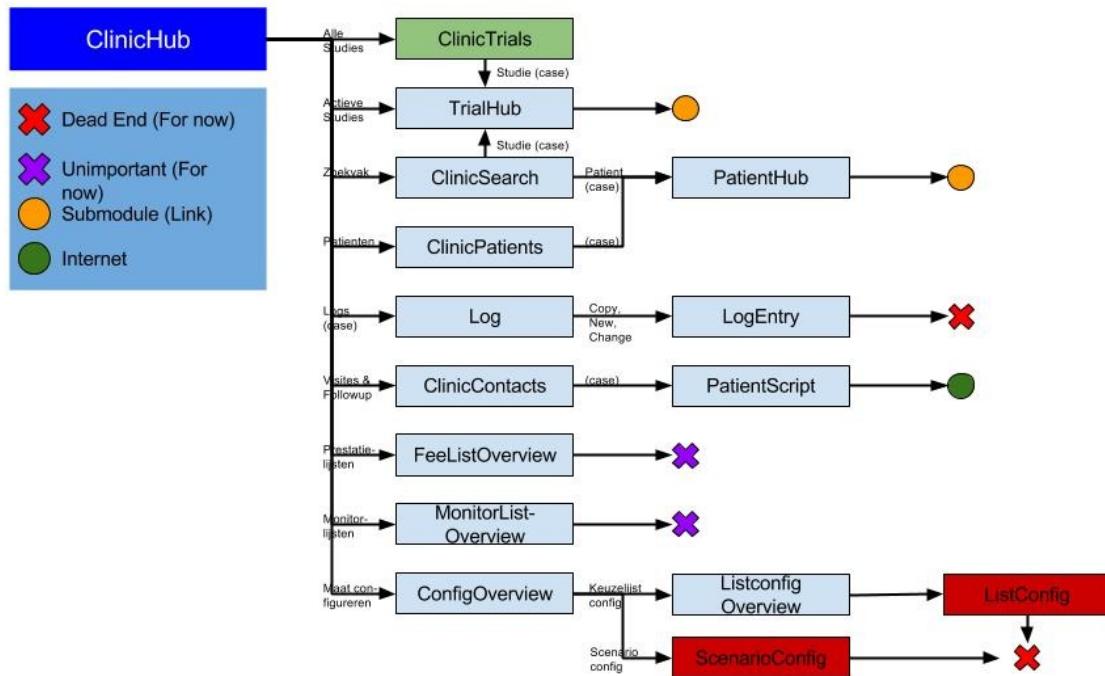


## 2.3 Maät ontdekken

### 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 subpagina, 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 hubpagina een nieuw bestand gemaakt waarin dezelfde werkwijze gevuld 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 bevindt.
  - 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 bevindt 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
  - **Functionaliteit**
    - 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 opgeliist 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 efficiënt 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 paradigmachecklist 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 paradigmachecklist, 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 paradigmachecklist. 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 paradigmachecklist 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 gekopieerd 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 moet 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 object. 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 bevindt.

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 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 dat 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 verwachte 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 verwachte 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 ordeningen
- 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 bevindt 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 geïmplementeerd 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 geanalyseerd 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 zijn 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 getimedde 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 onder andere 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 bevindt.

De tests zullen dus bestaan uit 2 delen, in het eerste deel wordt het scenario gecreëerd dat getest gaat worden (initieel, 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 getapt 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 overzichtelijker 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 onmiddellijk 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 geordend 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 eigenlijk 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 specieker. 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 specifieker 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 essentiële 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 namelijk 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-staat 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" ResultLabel="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 geannuleerd)
    - 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 genereert 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 genereert 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 weergeven 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 defineert een doel voor elke testcase. Het Target XML bestand bevat volgende syntax:

```
<Target TargetName="Description" CategoryId="Category GUID" ObjectId="Object GUID" TargetLabel="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 efficiënte 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 paradigma's 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 paradigma's inklaapbaar 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. Heir voor 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.

	ClinicHub	Menubar	ToggleButtonHome	ToggleButtonExpand	ToggleButtonSettings	AppWindow	ButtonBack	SearchBox	SemanticZoom	HubSectionAgreementInfo	HubSectionMyStudies	ListItemTrials	ButtonAlleStudies	HubSectionVisitesFollowUps	ListItemVolgensRaak	ListItemVolgensChronologie	HubSectionLogs	ListItemLogs	HubSectionActiveDocumenten	HubSectionAntreneTaken	ButtonPrestatielijsten	ButtonMonitorlijsten	ButtonMaatconfigureren	HubSectionZeeOok	ButtonPatients
AllParadigms	89	59	14	100	-	1	93	89	100	94	T	88	83	100	T	T	T	T	T	100	100	100	100	100	
Content	100	N	N	-	-	5	T	-	T	-	T	T	-	T	T	T	T	T	-	-	-	-	-	-	
CRUD	100	N	N	-	-	5	T	-	T	-	T	T	-	T	T	T	T	T	-	-	-	-	-	-	
Create	-	-	-	-	-	-	100	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
DataCreatedInDB	-	-	-	-	-	-	-	100	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
DeleteUpdate	N	N	N	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
DataModifiedInDB	N	N	N	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
DataModifiedInApp	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

# 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 opgeliist

- 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 Stores 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 gesynced 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 installeren
        - 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 verificatiestests) 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.csv", "data#csv",
DataAccessMethod.Sequential), DeploymentItem("data.csv"), TestMethod]
public void ClickPatientSearchResults()
{
    // set searchproperty to value in datafile
    UIPatiëntenHubSection.UIPatiëntenText.
    UIItemList.UICalidosMaaatClientLogListItem1.SearchProperties[XamlControl.PropertyNames.Instance] = TestContext.DataRow[0].ToString();

    // create control based on new searchproperty
    XamlControl Control = new XamlControl(UIMap.UICalidosMaaatWindowsWindow.UISemanticZoomSemanticZoom
    .UIPatiëntenHubSection.UIPatiëntenText.UIItemList.UICalidosMaaatClientLogListItem);

    //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-staat)**

### **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 versturen 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 tegen gekomen 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 Separated 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/](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/dpkssinghal/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](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/](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".

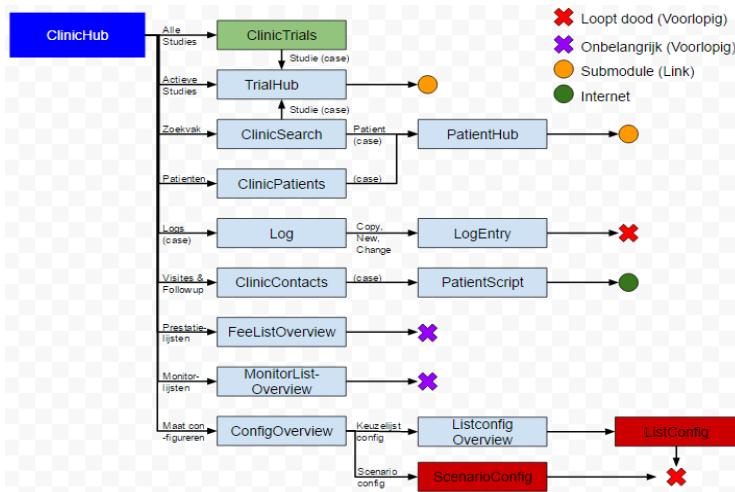
\*\*Deze documenten zijn niet aanwezig in de scriptie omdat het format dit niet toelaat.

# Testing Guideline

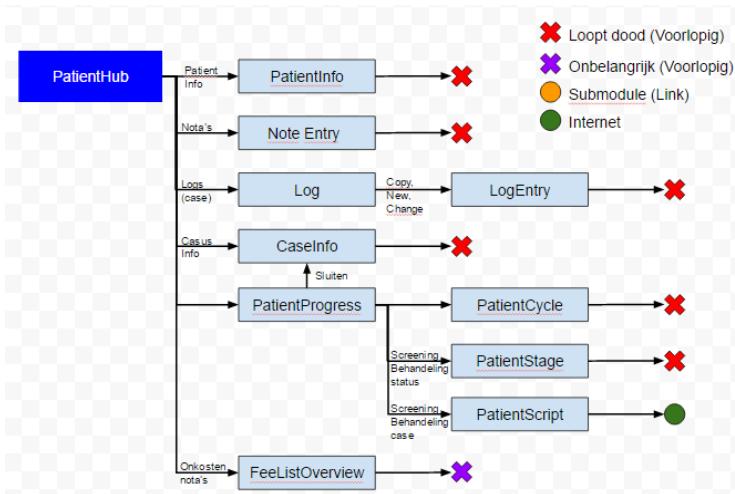
## Hierarchical navigation design

In this section, you can see a hierarchical design of all possible navigations inside Maät, ordered by page. For every page, we have one UI-map and per page, we have a test project for every paradigm on the checklist.

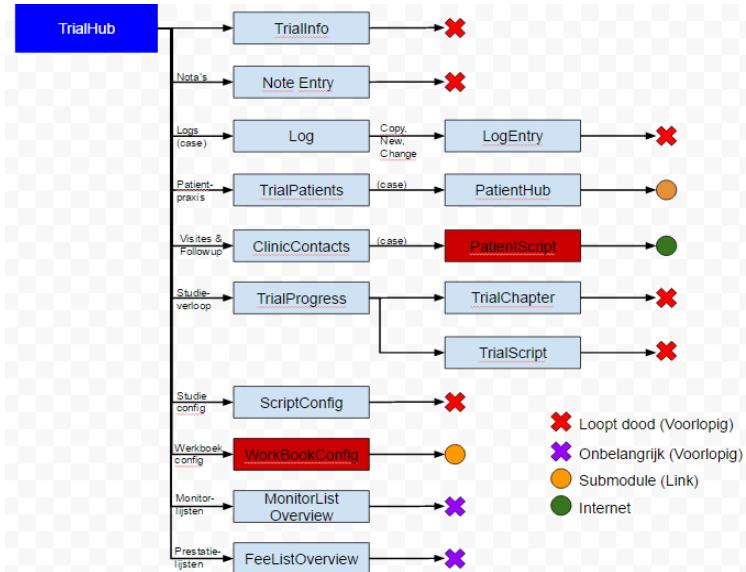
### ClinicHub



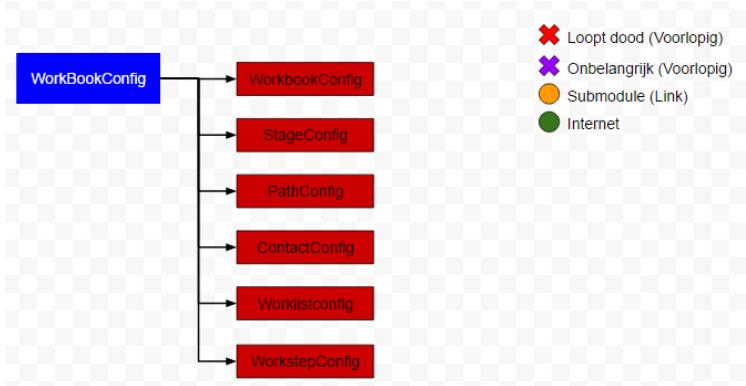
### PatientHub



## TrialHub



## WorkBookConfig



## Method Naming

To maintain the readability of the test project, here are some guidelines on how to name your test methods

Page name + Paradigm (+ Context) (+ Xaml Name Property) (+ optional info)

- Ex: ClinicSearchNavigateStudieListItemState1()
- Page name: ClinicSearch
  - Page name indicates the page within which your test is situated
- Paradigm: Navigate
  - Paradigm is the kind of action you are testing for the control/set of controls
  - Ex: navigate, zoom, toggle,...
- Context: Studie
  - Indicator of the context in which the control/set of controls you want to test exist, in this case, the method tests a "Studie-searchresult"
- XamlNameProperty: ListItem
  - In case of testing one specific control (or set of the same controls), this will indicate the kind of controls tested
- Optional info: State1
  - this will indicate extra info about the context of your test, in this case, the extra info indicates the stage of the page.

## File Structure

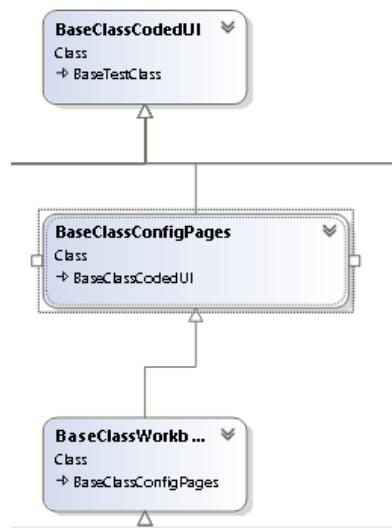
Maät follows a specific map structure for all it's elements. To keep the file structure readable, we follow the same structure inside our test project. For more information on the hierarchy, refer to the next page.

Calidos.Maat.CodedUITests

- Screens
  - General UIMap
  - BaseClassCodedUI partial classes
  - .csv global scenarios
  - Category (clinic,...)
    - Optional BaseClass per category
    - Optional UIMap per category
    - .csv categorical scenarios
    - Page (contactspage,...)
      - Ulmap per page
      - BaseClass per page
      - .csv local scenarios
      - .cs files per paradigm (content, navigations,...)

## Testproject hierarchy

In the progress of our project, we often found that some functions could write some functions in parent-classes to make them reusable. The end result of this is the following hierarchy:



BaseTestClass is the overall class for the entire test project. This class has been written to enable the possibility of implementing test functions on different test project.

BaseclassCodedUI inherits from BaseTestClass, and contains all the functions for testing "Maät".

BaseClassCategory (in this case the ConfigPages) is a baseclass containing global functions and variables per screen-category. This is an optional class for those pages that are visually very similar, it inherits from the BaseClassCodedUI.

BaseClassPage (in this case WorkBookConfig) inherits from BaseClassCategory and contains all private functions and variables used to test one page. All test projects per paradigm inherit from this class.

## Checklist: Testing

This is the checklist to review for every page. This means that for every page in the application, we need to test every paradigm documented in the checklist (control per control). If we find new paradigms while analyzing new pages, we will add these to the checklist and review all previously tested pages for the newly found paradigms.

This document contains the full checklist in which every paradigm that needs to be tested is fully explained together with how to test this paradigm. We also have an Excel document which contains the following sheets:

### General checklist

In this sheet, the general paradigms are displayed and we have indicate for every page if this paradigm is already tested or not. It looks like this:

Workpoints	Circumstances	Contextual	Condition	Content	Control	Customization	Editing	Feedback	Formatting	Localization	Navigation	Paradigm	Scripting	Structure	Workflow	Workstation	Feelings	Motivation	Sensory	Log	Memory	Notability	Character	Perception	Performance	Positioning	Planning	Principles	Training	Triggers	Triggers												
<b>1. CONTENT</b>																																											
1.1 Create																																											
1.2 Read	E	D	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O															
1.3 Update/Delete	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N															
1.4 Speed	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O															
1.5 Custom	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N															
<b>2. NAVIGATION</b>																																											
2.1 Navigate to correct page	E	L	O	L	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	P	O	O	O	O	O	O	O															
2.2 Check correct page state	E	Y	O	N	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	P	O	O	O	O	O	O	O															
<b>3. STATE</b>																																											
3.1 Semantic Zoom	D	D	D	N	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	D	O	O	O	O	O	O	O	O														
3.2 Overlay	N	N	N	Y	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O	O	O	O	O														
3.3 Filtering search	D	N	N	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O	O	O	O	O														
3.4 Multiselect	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N														
<b>4. FUNCTIONALITY</b>																																											
4.1 Scrolling	D	L	O	D	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	P	O	O	O	O	O	O	O	O														
4.2 Control State Verification	D	E	O	P	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	P	O	O	O	O	O	O	O	O														
4.3 Control Accessibility	D	D	D	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O	O	O	O	O														
4.4 Custom	D	L	O	L	D	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	N	O	O	O	O	O	O	O	O														
<b>LEGENDA</b>																																											
<table border="1"><thead><tr><th>YES</th><th>TO DO</th><th>IN PROGRESS</th><th>LATER</th><th>DOES NOT APPLY</th><th>OPEN</th><th>ERROR</th></tr></thead><tbody><tr><td>3 items</td><td>21 items</td><td>5 items</td><td>6 items</td><td>68 items</td><td>523 items</td><td>4 items</td></tr></tbody></table>																														YES	TO DO	IN PROGRESS	LATER	DOES NOT APPLY	OPEN	ERROR	3 items	21 items	5 items	6 items	68 items	523 items	4 items
YES	TO DO	IN PROGRESS	LATER	DOES NOT APPLY	OPEN	ERROR																																					
3 items	21 items	5 items	6 items	68 items	523 items	4 items																																					

## Paradigm checklist

In the Paradigm checklist, we have a more detailed list for the checklist, containing all the subcategories explained in this document. Here, we indicate whether a certain paradigm is already analysed on the "How to test?"-aspect. It looks like this:

↓ WORKPOINTS	
<b>1. CONTENT</b>	
<b>1.1 CREATE</b>	
Check if "Data" is added to "Database"	Y
<b>1.2 READ</b>	
<b>1.2.1 Search algorithm</b>	
Check all possible search parameters the algorithm uses	L
Check if all searchresults contain the searchword	Y
Check if all possible searchresults are displayed	L
<b>1.2.2 Displayed data</b>	
Check if "Data" in control equals "Database data"	Y
<b>1.3 UPDATE/DELETE</b>	
Check if "Data" in "Database" is modified	L
Check if "Data" is changed in the "Application" where it should have changed	Y
<b>1.5 CUSTOM</b>	
Check page specific functionality dependent of "Data"	L
<b>2. NAVIGATIONS</b>	
2.1 Check if we navigated to the right page when control is activated	Y
2.2 Check if navigation triggers the correct page-state	Y
<b>3. STATES</b>	
<b>3.1 SEMANTICZOOM</b>	
<b>3.1.1 Zoom out</b>	
Select semantic zoom & use "Ctrl-"	Y

## Example page + Page checklists

In the example page, we have the same list as in the Paradigm checklist, but on top of it, but together with some of the basic controls which are present on every single page. On the Page checklists (one for each page), we use the example page and then search for every possible testable control. For every control we mapped, we indicate whether a certain paradigm needs to be tested on that control or not. The result looks like this:

PARENT CONTROL NAME →		Menubar			App Window			
PARENT CONTROLYTYPE →		Window		Window				
WORKPOINTS	CONTROLYTYPES →	Home	Expand	Setting	Back	Button		
<b>1. CONTENT</b>								
1.1 CREATE								
Check if "Data" is added to "Database"								
1.2 READ								
1.2.1 Search algorithm								
Check all possible search parameters the algorithm uses								
Check if all searchresults contain the searchword								
Check if all possible searchresults are displayed								
1.2.2 Displayed data								
Check if "Data" in control equals "Database data"								
1.3 UPDATE/DELETE								
Check if "Data" in "Database" is modified								
Check if "Data" is changed in the "Application" where it should have changed								
1.5 CUSTOM								
Check page specific functionality dependent of "Data"								
<b>2. NAVIGATIONS</b>								
2.1 Check if we navigated to the right page when control is activated								
2.2 Check if navigation triggers the correct page-state								
<b>3. STATES</b>								

The first small column is a summary of each row, to indicate whether this paradigm is tested for all controls it needed to be tested for.

PARENT CONTROL NAME →		Menubar			App Window			commands	
PARENT CONTROLYTYPE →		Window			Window				
WORKPOINTS	CONTROLYTYPES →	Home	Expand	Setting	Back	Alle studies	Alle open studies	ClinicTrialsHub	Nieuwe studie s
<b>CONTENT</b>									
CREATE									
Check if "Data" is added to "Database"									
READ									
Check if "Data" in control equals "Database data"									
UPDATE									
Check if "Data" in "Database" is modified									
Check if "Data" is changed in the "Application" where it should have changed									
DELETE									
Check if "Data" in "Database" is deleted									
CUSTOM									
Check page specific functionality dependent of "Data"									
<b>NAVIGATIONS</b>									
Check if we navigated to the right page when control is activated									
Check if navigation triggers the correct page-state									
<b>STATES</b>									
SEMANTICZOOM									

## **1. Content**

This section contains every possible test scenario, which involves the data which is loaded in the application. Note that some content tests are still performed hardcoded (we assert to a hardcoded value).

### **1.1 CRUD**

Crud tests involve every action we can perform on the data in the user interface. Crud is an abbreviation for Create, Read, Update and Delete.

#### 1.1.1 Create

what?

Check if adding data (for example adding patient) and clicking the execute button adds the data you added to the database, do this by asserting the elements in the user interface where this data should be added after clicking the execute button.

how?

1. Add all textfields (and if necessary, the "execute" button) involved in the "create"- action, to the UIMap. If the datafields aren't accessible through normal UI-mapping, add the parent control you can access to the UIMap and use the technique of getting children to reach the control's you want to use.
2. In case of textfields, clear all textfields with the "ClearEdit(textfield)" function from the BaseClassCodedUI.
3. Add content to the textboxes by using the following line of Code:

```
TextBox.Text = content;
```

in which *content* is replaced by the data you want to add. To do this with external data: go to "How to test" - "Test Methods" - "The datasource tag".

4. If necessary, click the execute button.
5. Navigate to the page where your input should have influence and execute a "Read" test on the influenced controls, to make sure the data is added. How to execute a "Read" test is explained in the next paragraph.

#### 1.1.2 Read

What?

Check if all controls on the page you are testing, containing data from the database, display the correct data for the parameters by which you got to that page. So for example, if you navigated to a trial with trial-id "4". All controls in the TrialHub should contain the data of trial "4", and not the data of for example trial "6" or completely faulty data.

How?

1. Execute the correct actions to go to the page you are testing, using variables such as for example "Case ID"/"Trial ID". Ideally, you should get this variable from a datafile in which all other data of this Case/Trial/... is available. For data driven tests: go to "How to test" - "Test Methods" - "The datasource tag".
2. Add the involved controls on the page to the UIMap.
3. To check if the data is correct, just assert to `control.Name` or `control.Text`, depending on the type of control involved. The expected value should be in your datafile, so to access this, check out data driven tests. If you do not use a data driven test, you can still work with hardcoded expected values.

#### 1.1.3 Update/Delete

what?

This paradigm is similar to create, but now you change already existing data instead of creating new data.

How?

The "how to" of Update/Delete is similar to the create.

1. Add all textfields (and if necessary, the "execute" button) involved in the "update"- action, to the UIMap.
2. Clear all textfields with the "ClearEdit(`textfield`)" function from the `BaseClassCodedUI`.
3. Add content to the textboxes by using the following line of Code:

`TextBox.Text = content;`

in which `content` is replaced by the data you want to add. To do this with external data: go to "How to test" - "Test Methods" - "The datasource tag".

4. If necessary, click the execute button.

Navigate to the page where your input should have influence and execute a "Read" test on the influenced controls, to make sure the data is updated. How to execute a "Read" test is explained in the previous paragraph.

## 1.2 Custom-CRUD

Under this sub-section, we will place all special Crud-functions.

### 1.2.1 General

If you want to make sure a certain word/value is displayed in every single ListItem in the list, you can use the ReadListItems-function:

```
ReadListItems( list, expectedValue);
```

This function loops over every ListItem and checks all datafields inside of it, if the ExpectedValue isn't displayed in one of the ListItems, the test will fail.

### 1.2.2 Search algorithm

What?

When using a SearchBox, you want to check 3 things.

- A. Check all the possible parameters on which you can perform a search (for example, search by name, search by trial, search by trial number,...)
- B. Check if all the search results contain the parameter you searched
- C. Check if all possible search results available in the database, based on your search, are all displayed

How?

- A. to do
- B. Use ReadListItems-function.

to do

## 2. Navigations

### 2.1 Correct navigation

What?

When navigating to a certain page, from the page you are testing, you have to make sure the application navigates to the correct page according to the navigation-control you used. To keep the testing project easily readable, we will consider each different tab of a page (if there are tabs available) as a different page.

How?

Variable Controls

A variable Control is a Control which is embedded in a list, in which all cases of a certain object are displayed. For example, you can have a list of trials, patients, logs,... In other words, "variable" in this context means that there is never a fixed amount of Controls in the list, and the Controls displayed, never have a fixed instance inside the list. These kind of Controls all navigate to the same page, but the data displayed on that page is dependent on the instance you navigated to.

To test the navigation of this kind of Control, use the following function:

```
NavigateVariableControl(parameters);
```

For more info about this function and its parameters, refer to "How to test"

- "BaseClassCodedUI"

## Fixed Controls

A fixed Control is a Control which is embedded in a HubSection, in which every possible navigation-control has a fixed instance. Also, there is a fixed amount of navigation-control's inside the hub. These kind of Controls normally navigate to a different page, or to a different state of the same page(for that, refer to the next paragraph).

To test the navigation of this kind of Control, use one of the following functions:

- `NavigateFixedControlToPage(parameters);`
- `NavigateFixedControlToTab(parameters);`

For more info about these function and their parameters, refer to "How to test" - "BaseClassCodedUI".

## Search navigation

- add the SearchBox to the UIMap
- add a searchword to the SearchBox with "TextBox.Text = ..."
- Hit enter
- Assert correct navigation by checking the title of the page navigated to

## Hyperlink navigation

Use the following function of the BaseClassCodedUI:

"`NavigateHyperLink(hyperlink,titlecontrol);`"

## 2.2 Correct page state

### what?

On some pages, you can use multiple controls to navigate to the same page. But every control triggers a different page state on the page you navigated to. So you need a test which can assert if the navigation-control you used, triggers the correct page-state on the page navigated to.

### How?

## Filtering search

In this case, the Controls trigger a certain FilterBox-state on the page navigated to. This is how to test this:

1. Add the control you want to navigate with to the UIMap
2. Generate an `ExpectedValue`, refer to "How to test" to see how to do this.
3. Navigate
4. Get the combobox involved in the filtering, and get its `SelectedItem`
5. Assert the `ExpectedValue` to the `SelectedItem`'s value

## 2.3 Load speed

What?

Check how fast a page is loaded when navigated to. Save this loading time and report.

How?

1. Add the progressbar to the UIMap, to this by using the “Ctrl-i”-key combination while the page is loading, hovering over the progressbar. You will sometimes have to try this a few times because it disappears quickly.
2. In your code, navigate to the page and start a StopWatch (more info about the stopwatch in “how to test”)
3. Write a while function that keeps running as long as the ProgressBar’s coördinates are not below 0.
4. After the while function, save the StopWatch.Elapsed property in a variable

Report the variable (still to analyse)

Met opmerkingen [1]: still to finish

## 3. States

“States” are all the different ways a page can be ordered and displayed. There are 4 different possible states.

- Overlay

Overlay is a state in which a partial popup window is displayed on top of the normal window of a page. This overlay has data-fields in it, that are intended to add or modify data in the application.  
An overlay looks like this:

- No overlay:

02Monaleesa28 Nieuw; Wervingreserve open	Monaleesa04 Nieuw; Wervingreserve open	Monaleesa08 Nieuw; Wervingreserve open	Monaleesa12 Nieuw; Wervingreserve open	Monaleesa16 Nieuw; Wervingreserve open
ArtsviewTest Aan de gang; Open voor inclusie	Monaleesa05 Nieuw; Wervingreserve open	Monaleesa09 Nieuw; Wervingreserve open	Monaleesa13 Nieuw; Wervingreserve open	Monaleesa17 Nieuw; Wervingreserve open
DatadrivenCUI12 Aan de gang; Wervingreserve open	Monaleesa06 Nieuw; Wervingreserve open	Monaleesa10 Nieuw; Wervingreserve open	Monaleesa14 Nieuw; Wervingreserve open	Monaleesa18 Nieuw; Wervingreserve open
Monaleesa03 Nieuw; Wervingreserve open	Monaleesa07 Nieuw; Wervingreserve open	Monaleesa11 Nieuw; Wervingreserve open	Monaleesa15 Nieuw; Wervingreserve open	Monaleesa19 Nieuw; Wervingreserve open

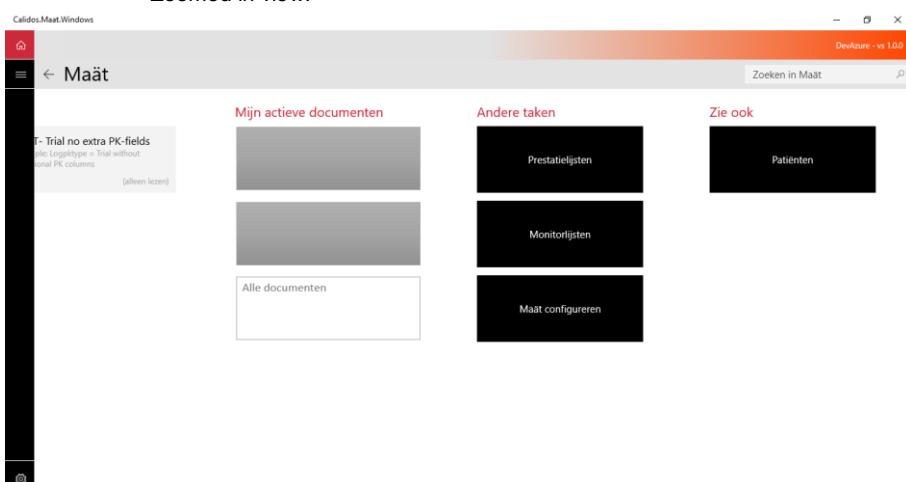
- Overlay:



- Zoomed in & zoomed out (semantic zoom)

The semantic zoom is a control within a page, which contains a hub. This hub contains different hubsections, each with it's own title. The semantic zoom is able to zoom out these hubsections so that they are displayed as a list of ListItems, each containing the text of one of the HubSection titles. This is a functionality that needs to be tested.

- Zoomed in view:



- Zoomed out view:



- Page filtering

"Page-filtering" contains all the different states of ordering a page can be in. For example, some elements in the page can be ordered by different factors like date, name,... Every different ordering state is a page filtering state

ex:

- By task filtering

Vandaag	Morgen	Te Crf'en	Vandaag te Crf'en
000P011 T00P011 in T001 Nihil Week 1, d -5 Tijdelijk Contact		T00P011 T00P011 in T001 Weeks1 1, d 1 C1	TestPatient UnitTestMedishePDF... C1 2, C2, d 1 Cyclus 1 Dag 1 Vandaag te CRF'n 13/03/2015 Joske Vermeulen in Monaleesa 28 Screening 2, C2, d 24 Screening d -7 Te CRF'n 22/04/2016
		Joske Vermeulen in Monaleesa 28 Screening 2, C2, d 24 Screening d -7 Te CRF'n 30/04/2016	Joske Vermeulen in Monaleesa 28 Screening 2, C2, d 24 Screening d -7 Vandaag te CRF'n 24/03/2016 Joske Vermeulen in Monaleesa 28 Am1A 3, C2, d 1 Anthracycline Vandaag te CRF'n 24/03/2016
			Joske Vermeulen in Monaleesa 28 Am1A 2, C2, d 1 Anthracycline Vandaag te CRF'n 24/03/2016

- By chronology filtering

- Multiselect

Multiselect is a state in which multiple elements can be selected. This state is not present inside any "Maät"-page, but it's a state that would be possible inside a Windows 10 UWP-application, so we put it in our checklist.

### 3.1 No overlay state

#### 3.1.1 Zoomed in state

Zoom out with Ctrl-

*what?*

Check if using the "ctrl" key in combination with the "-" key makes the semantic zoom zoom out. Note that the semantic zoom needs to be selected before this action is possible.

*How?*

1. Add the hubsection you want to zoom out to, to the UIMap
2. Add the name of the hubsection to the UIMap
3. Add the zoomed out listitem, which represents the hubsection, to the UIMap
4. Use following function:  
`AssertSemanticZoom(hubsection, hubsectionSemanticName, Action.CtrlKey);`

Zoom out with Ctrl & mouse scroll

*what?*

Check if using the "ctrl" key in combination with the "mousewheel" makes the semantic zoom zoom out. Note that the mouse needs to be hovered over the semantic zoom before this action is possible.

*how?*

Do the same as the above method, but use "Action.CtrlScroll" instead of "Action.CtrlKey".

Zoom out with Tab/arrow select

*what?*

Check if using the tab key (or arrow keys) selects the HubSection titles in the semantic zoom and using the -

- space
- enter

-key makes the semantic zoom zoom out.

*how?*

**//PROBLEM: Cannot read whether the HubSection-titles are selected or not.**

Zoom out with PgUp/PgDn select

*what?*

Check if using the PgUp/PgDn keys selects the HubSection titles in the semantic zoom and using the -

- space
- enter

-key makes the semantic zoom zoom out.

*how?*

**//PROBLEM: Cannot read whether the HubSection-titles are selected or not.**

Zoom out with Mouseclick

*what?*

Check if clicking on a hubsection-title makes the semantic zoom zoom out.

*how?*

AssertSemanticZoom(Hubsection, HubsectionSemanticName, HubsectionTitle);

- Hubsection: Hubsection you want to test
- HubsectionSemanticName: Name control of zoomed out listItem representing hubsection
- HubsectionTitle: Title control of hubsection in zoomed in view

Check hubsection content

*What?*

Check if correct data is displayed in hubsections when zoomed out:

- Checkboxes
- hubsection titles

On some pages, every element in the semantic zoom contains a checkbox, when zoomed out, the checkboxes on the zoomed out listItems have to be in the same state as the checkboxes on the zoomed in view.

example:

How?

- Checkboxes:

1. Add both lists (zoomed in and zoomed out view) to the UIMap. Sometimes these lists do not contain a UID, so make sure to change the searchproperties afterwards. To make sure your test method always finds the correct list, add the correct instance to the list (in the above case, both lists have instance "2").
2. Add the semantic zoom to the UIMap
3. Get all the elements of the list by using following function of the baseclassCodedUI:

```
UITestControlCollection ZoomedInGroups = getHubSections(List);
```

4. Make a bool-array to store the content of all checkboxes, give it the length of the UITestControlCollection you just created
5. Store every checkboxes state in the array with a for function that loops over every element in your collection. The checkbox can be found by using the "GetChildren()" function several times. To know which instance to use and how deep you have to search in the hierarchy of your collection, use the CodedUITestBuilder to see the place of the desired checkbox inside the hiérarchy. In the case above, the checkbox can be found by getting the second child of every element in the collection, and getting the first child of that child. To store the state of the checkbox, you have to use the ".Checked"-property.
6. Zoom out using the "ZoomOutSemanticZoom(Action.CtrlScroll, SemanticZoom);"
- function.
7. Make a new UITestControlCollection for the zoomed out list using the same method as before.
8. Assert the lengths of both collections (should be the same!)
9. Assert the state of the zoomed out checkboxes is equals the bool-array you created earlier, with a for loop, that loops over every zoomed out ListItem. To find these checkboxes, use the same technique as described above (with GetChildren())
10. Zoom in the semantic zoom using following function:  
`ZoomInSemanticZoom(ZoomedOutListItems[0] as XamlControl, Action.CtrlScroll, SemanticZoom);`

- Hubsection titles

To check if the titles displayed in the zoomed out list of the HubSections, are the same as the titles in the zoomed in state (and the same amount), use the following function of the BaseClassCodedUI:

`CheckSemanticZoomTitles()`

*note: this only works when the SemanticZoom contains a Hub with HubSections, if the elements of the SemanticZoom are different, you have to make a custom-function.*

### 3.1.2 Zoomed out state

temporary for Zoom out & Zoom in: BaseClassCodedUI function:

`AssertSemanticZoom(Hubsection, HubsectionSemanticName, action);`

- Hubsection: Hubsection you want to test
- HubsectionSemanticName: Name of zoomed out listItem representing hubsection
- action: CtrlKey/Scroll

Zoom in with Ctrl+

*what?*

For each page, check if the semantic zoom functionality of the page zooms in when using the “Ctrl” key in combination with the “+” key. Check if the selected hubsection is positioned correctly (on the left side of the screen). To select a different hubsection, use the tab key.

*how?*

5. Add the hubsection you want to zoom out to, to the UIMap
6. Add the name of the hubsection to the UIMap
7. Add the zoomed out listitem, which represents the hubsection, to the UIMap
8. Use following function:

```
AssertSemanticZoom(hubsection, hubsectionSemanticName, Action.CtrlKey);
```

Zoom in with Ctrl and mouse scroll

*what?*

For each page, check if the semantic zoom functionality of the page zooms in when using the “Ctrl” key in combination with the “mousewheel”. Check if the selected hubsection is positioned correctly (on the left side of the screen). To select a different hubsection, use the tab key.

*how?*

Do the same as the above method, but use “Action.CtrlScroll” instead of “Action.CtrlKey”.

Zoom in with Tab/arrow select

*what?*

Check if using the tab key and arrow keys selects the HubSection titles in the semantic zoom and using the -

- space
- enter

-key makes the semantic zoom zoom in.

Check if the selected hubsection is positioned correctly (on the left side of the screen).

*how?*

Zoom in with PgUp/PgDn select

*what?*

Check if using the PgUp/PgDn keys selects the HubSection titles in the semantic zoom and using the -

- space
- enter

-key makes the semantic zoom zoom in.

Check if the selected hubsection is positioned correctly (on the left side of the screen).

*How?*

Zoom in with Mouseclick on hubsection-titles

*what?*

For every page, check if the semantic zoom functionality of the page zooms in when clicking on the titles of different hubsections. Check if the clicked hubsection is positioned correctly (on the left side of the screen).

*how?*

`AssertSemanticZoom(Hubsection, HubsectionSemanticName, HubsectionTitle);`

- Hubsection: Hubsection you want to test
- HubsectionSemanticName: Name control of zoomed out listItem representing hubsection
- HubsectionTitle: Title control of hubsection in zoomed in view

### 3.1.3 Filtering search

- Check if Combobox ordering function orders the page correctly

**//NOT YET ANALYSED//**

### 3.1.4 Multi Select

Multiselect is a state in which you can select multiple items at once. The application Maät does not contain this state, but it is a possible state in a windows 10 metro application and needs to be tested if it's there.

### 3.2 Overlay state

#### Open overlay button

- What?

Check if using the button intended to open the overlay, triggers the overlay appearance.

- How?

1. Add button to the UIMap
2. Add the full overlay to the UIMap as a control (this will be a Hub)
3. Click the button
4. Assert on the coordinates of the hub, if the hub is visible, these should have a value above "0". Otherwise, all values should be "-1".

*note: The overlay needs to be opened and used by your test method, before it can see the coördinates being set to -1 after closing it. If you don't open it initially and try to search its coördinates, the test method will fail to find the overlay in the first place.*

#### Close overlay button

- what?

Check if using the button intended to close the overlay, makes the overlay disappear.

- How?

1. Add "x" button to the UIMap (for some overlays, you may need to give an automationId to the button).

ex.

```
<!-- CANCEL -->
<AppBarButton x:Name="CloseOverlayButton"
```

This line is located in the ClinicTrials.NewTrial - page, in which NewTrial is the overlay of ClinicTrials.

2. Add the overlay hub to the UIMap
3. Use the XamlControl.TryFind() function to make sure the overlay is opened.
4. Click the button & add delay to make sure the overlay has time enough to close completely
5. Check if one of the coördinates of the OverlayHub is still larger than zero, normally when a control is not visible on a page, it's coördinates will all be set to "-1".

```
public void ClinicTrialsPageOverlayClickCloseOverlayButton()
{
    XamlControl OverlayHub = UIMapClinicTrials.UICalidosMaatWindowsWindow.UIEnnieuwstudietoevoHub;
    XamlControl CloseButton = UIMapClinicTrials.UICalidosMaatWindowsWindow.UICloseOverlayButton;
    OverlayHub.TryFind();
    UseControl(CloseButton, Action.Click);
    Playback.Wait(delayMilliseconds);
    if (OverlayHub.Left > 0) Assert.Fail();
}
```

#### Close overlay with "esc"

- What?

#### Check if overlay closes when pressing the "esc"-key

- How?

Same method as above, but instead of clicking a button, use Keyboard.SendKeys("{Escape}")

### Enable execute button

- What?

Check if "execute" button is only enabled when correct data fields are filled in

- How?

1. Make sure the data fields involved are all empty before you begin checking the button's state.
2. Check button's state after filling in datafields

ex.

```
public void ClinicTrialsPageOverlayCheckStudieToevoegenButtonState()
{
    ClearEdit(StudieNummer);
    ClearEdit(ProtocolNummer);
    ClearEdit(Alias);
    if (ToevoegenButton.Enabled) Assert.Fail("Button is enabled when it should be disabled!");
    StudieNummer.Text = "Blabla";
    if (ToevoegenButton.Enabled) Assert.Fail("Button is enabled when it should be disabled!");
    ProtocolNummer.Text = "Blabla";
    if (ToevoegenButton.Enabled) Assert.Fail("Button is enabled when it should be disabled!");
    Alias.Text = "Blabla";
    if (!ToevoegenButton.Enabled) Assert.Fail("Button is disabled when it should be enabled!");
}
```

## 4. Control state appearance

In this section, we summarize all tests that check the visual appearance or state of a control. This includes things as the color, the fact that it is highlighted, the fact that it is enabled to be used, the fact that the mouse cursor changes when hovering over or clicking the control,...

### 4.1 Initial

what?

Check if the control's initial state (appearance) is as it should be.

how?

color

The displayed color of the control.

**/TO ANALYSE**

*highlighted*

When a control is selected (ex through tab) we call it "highlighted"

To test this, assert to the controls "HasFocus"-property, which can be true or false.

*enabled*

"Enabled" means the control's function are available to be used, when it's disabled, the control is displayed but cannot be used.

1. Create the correct conditions for the control to be enabled/disabled and then assert to the "Control.Enabled"-property, which is a boolean.

*Placeholder text*

The default text is the text you sometimes see in a textbox that disappears as soon as you start typing. It's a text indicating the type of data you have to put in the textbox (bvb "name")

**/TO ANALYSE**

*default value*

The initial value displayed by a control

Assert to the controls "Text" or "Value" property.

4.2 Hovered

What?

Check if the control's state changes correctly when hovering over the control.

How?

Use "Mouse.Hover" to hover over the desired control.

*Mouse/cursor change*

**/TO ANALYSE**

*color*

**/TO ANALYSE**

*highlighted*

To test this, assert to the controls "HasFocus"-property, which can be true or false.

*enabled*

assert to the "Control.Enabled"-property, which is a boolean.

*Placeholdertext*

**/TO ANALYSE**

*default value*

Assert to the controls "Text" or "Value" property.

4.3 Clicked

What?

Check of the control's state changes correctly when clicking on the control.

How?

Use "Mouse.Click" or the baseclass function "UseControl" to click on the desired control.

*Mouse/cursor change*

**/TO ANALYSE**

*color*

## //TO ANALYSE

*highlighted*

To test this, assert to the controls "HasFocus"-property, which can be true or false.

*enabled*

assert to the "Control.Enabled"-property, which is a boolean.

*PlaceholderText*

## //TO ANALYSE

*default value*

Assert to the controls "Text" or "Value" property.

## 4.4 Click & hold

What?

Check if the control's state changes correctly when clicking on it and holding your click.

How?

Use "Mouse.StartDragging" function to press the left mouse button and holding it down.

*Mouse/cursor change*

## //TO ANALYSE

*color*

## //TO ANALYSE

*highlighted*

To test this, assert to the controls "HasFocus"-property, which can be true or false.

*enabled*

assert to the "Control.Enabled"-property, which is a boolean.

*PlaceholderText*

## //TO ANALYSE

*default value*

Assert to the controls "Text" or "Value" property.

*Control deepens?*

This means that the control will display a subtle animation when clicked, which in this case is that the control moves deeper into the screen.

## //TO ANALYSE

#### 4.5 Filled in

What?

Check if the control's state changes correctly when filling it in.

How?

Fill the control by changing its "Text" or "Value" property.

*color*

*highlighted*

To test this, assert to the controls "HasFocus"-property, which can be true or false.

*enabled*

assert to the "Control.Enabled"-property, which is a boolean.

*value*

Assert to the controls "Text" or "Value" property.

### 5. Functionality

In this section, we will discuss how to test all the possible functionalities a control can have, other than navigating to another page.

#### 5.1 General

These are the general functionalities of every control, which include its selectability by using tab or arrows, as well as its ability to execute its function with a click, enter or space action. Some controls will be usable with enter/space while other don't. The same is true for its selectability.

##### Click selection

What?

Check if the control is selectable with a click action.

how?

1. Click on the control
2. assert on the "control.HasFocus" property. If the control is selected, this property should be true.

##### tab & arrow selection

what?

Check if the control is selectable through tabbing and using the arrow keys

how?

1. Go to the control by using the BaseClass-functions TabToControl() and GoToControl()
2. assert on the "control.HasFocus" property. If the control is selected, this property should be true.

### Click execution

what?

check if the control can be used with click action.

how?

1. Click on the control
2. Assert to the action the control was supposed to execute.

### Enter execution

what?

check if the control can be used with enter action

how?

1. Make sure the control is selected
2. Hit enter
3. Assert to the action the control was supposed to execute.

### Space execution

what?

check if the control can be used with space action

how?

1. Make sure the control is selected
2. Hit space
3. Assert to the action the control was supposed to execute.

## 5.2 Execution functionality

In this part we will discuss every possible function a control can have. In this section it does not matter whether we use click/enter/space, but it matters what happens when we use one of these actions.

### Partpicker data selection with arrow-keys

what?

check if the partpicker section changes the number with up and down arrow keys (when selected)

how?

**//PROBLEM: cannot access the popup window, nor the different tabs inside of it through code. Only the checkmark and cross button are accessible.**

### Partpicker data selection with scroll wheel

what?

check if the partpicker section changes the number with the scroll wheel (when selected)

how?

**//PROBLEM: cannot access the popup window, nor the different tabs inside of it through code. Only the checkmark and cross button are accessible.**

Open popup window

what?

Check if using the control opens up a pop up window

how?

1. Add the popup-window to the UI Map by using the "Ctrl-i" combination
2. Use the control
3. Assert the popup-window exists with the coordinate properties, these should be set to -1 if the popup-window isn't visible. Sometimes the test will even fail when it's trying to find the popup-window when it's not visible.

Close popup window with control

what?

check if using the control closes the popup window

how?

1. Add the control to the UI Map by using the "Ctrl-i" combination
2. Use the control
3. Assert the popup-window is gone with the coordinate properties, these should be set to -1 if the popup-window isn't visible. Sometimes the test will even fail when it's trying to find the popup-window when it's not visible.

Close popup window by clicking elsewhere

what?

When a popup-window is opened, and you click elsewhere with the mouse, the popup-window should close automatically.

how?

1. Make sure the popup-window is opened
2. Click on a hard coded coordinate, that is outside of the popup-window
  3. Assert the popup-window is gone with the coordinate properties, these should be set to -1 if the popup-window isn't visible. Sometimes the test will even fail when it's trying to find the popup-window when it's not visible.

Close popup window with "esc"

what?

When a popup-window is opened, and you click elsewhere with the mouse, the popup-window should close automatically.

how?

4. Make sure the popup-window is opened
5. press the "esc"-key
6. Assert the popup-window is gone with the coördinate properties, these should be set to -1 if the popup-window isn't visible. Sometimes the test will even fail when it's trying to find the popup-window when it's not visible.

#### Check data change in control

what?

Sometimes, using a control involves the input of some data, and after executing, the control will display the data you put in. The test is to check if inputting the data and executing results in the data you put in, displayed in your control.

how?

1. Execute the data change action, which can be typing text, selecting partpicker numbers,...
2. Make sure you save the data you put in as a variable in your test
3. Perform a read-test on the control, looking for your saved variable

#### Control-visibility changing

what?

check if using the control makes the correct other control's appear or disappear

how?

1. Use the control
2. Assert to the coördinates of the controls that should appear or disappear

#### Control reactivity

what?

check if you can use the control before the page is fully loaded

how?

1. Use the control you want to check the reactivity on, as you would do it to check the functionality of the control. Make sure the code is written in such a way that the control is used as soon as possible.
2. Check if the progressbar on top of the screen is still visible after the previous code. If it isn't, write an Assert.Fail() function.

## Control scrolling functions

scroll wheel

*what?*

Check if the scroll function works with the "mousewheel"-action.

*How?*

Use the following function of the BaseClassCodedUI:

`ScrollToControl(control, scrollControl, scrollDirection)`

- control: The control you want to scroll to
- scrollControl: The control you want to start scrolling from
- scrollDirection: Up or Down

click and drag scrollbar

*what?*

Check if the scroll function works when clicking and holding the scrollbar and then dragging it across.

*How?*

**//PROBLEM: Can not access the scrollbar in any way other than manual**

click on scrollbar arrows

*what?*

Check if the scroll function works with when clicking the arrows of the scrollbar.

*How?*

**//PROBLEM: Can not access the scrollbar in any way other than manual**

Tooltip

*what?*

Check if hovering over the control makes a tooltip appear

*How?*

**//NOT YET ANALYSED**

### Toggle state

what?

Check if using the control changes it's toggle state

how?

1. Use the control
2. Assert on the *control.Pressed* property

### Clipboard functions

Add the clipboard popup menu to the UIMap by using "Ctrl-i"

Cut

what?

Check if choosing "cut" on selected text makes the selected content disappear but still available to paste elsewhere.

how?

1. Select the text inside the control (still to analyse how to do this)
2. Right click on the text and click on the clipboard function: cut
3. Assert the correct actions happened with the control's content by using the "*control.Text*"-property

Copy

what?

Check if choosing "copy" on selected text makes the selected content available to paste elsewhere.

how?

1. Select the text inside the control (still to analyse how to do this)
2. Right click on the text and click on the clipboard function: copy
3. Assert the correct actions happened with the control's content by using the "*control.Text*"-property

Paste

what?

Check if choosing "paste" makes content you cut or copied, appear in the control you chose to paste in. Make sure to check if the content that was already in the control stays in the control, unless you selected it first.

*how?*

1. Right click in the control and click on the clipboard function: paste
2. Assert the correct actions happened with the control's content by using the "control.Text"-property
3. Do the same as above but first select a part of the already existing content in the control
4. Assert to the content again

Undo

*what?*

Check if choosing "undo" makes the content inside the control you are working in disappear.

*how?*

1. Right click on the control
2. click on the clipboard function: undo
3. Assert the correct actions happened with the control's content by using the "control.Text"-property

Select all

*what?*

Check if choosing "select all" selects all content inside of the control you are working in

*how?*

1. Right click on the control
2. click on the clipboard function: select all
3. Assert the correct actions happened with the control's content by using the "control.Text"-property

Clear control's content

*what?*

Check if clicking the "x" in the control sets it back into default state

*how?*

1. Click on the "x"-button inside the control
2. Check if the control is cleared with the "control.Text"-property
3. Check if the control is back into default state by executing "Control state appearance" tests

Validation text

*what?*

Check if inputting the wrong content into the control triggers the appearance of a validation text, which is a red line of text with a warning message.

how?

Perform a “Control visibility changing” test after inputting the wrong content

#### Default selected control

what?

Sometimes, there is a control selected by default, without performing any further actions. Check if this control is selected when it should be.

how?

1. Create the correct conditions for the control to be selected
2. Assert to the “*control.HasFocus*”-property

#### Typing content into selected control

what?

Check if typing some content when your control is selected, results in the content you typed appearing inside the control you are testing

How?

1. Select the control
2. Use the “*WriteAsUser(content)*” function
3. Assert to the “*control.Text*”-property

#### Typing content randomly into control

what?

Sometimes, when typing randomly, the content you are typing will appear in a control by default. Check this functionality

how?

Do the same as above, but without selecting the control

#### Clear control with “Esc

what?

Some controls can be cleared by pressing the “esc” key. Test this functionality

how?

1. Make sure there is content inside the control you are testing.
2. Hit the “Esc”-key
3. Assert to the “*control.Text*”-property

### Search with magnifying glass

what?

Inside a SearchBox, there is a magnifying glass. By clicking it, you will execute the same search action as if you were hitting enter. Test this functionality.

how?

**//PROBLEM: magnifying glass is not accessible through any other way than manual**

### Select control-item with arrow keys

what?

Check if using the arrow-keys on a selected control (up and down-keys) changes the control's selected item

how?

1. Select the control with tab.
2. Press the arrows up or down.
3. Assert that the content of the control has changed correctly.

### Select control-item by clicking

what?

By clicking on the control, you open a dropdown-menu and can select an item inside of it. The drop down menu will disappear and your selected item will appear inside the control. Test this functionality.

how?

**//TO ANALYSE**

## 6. Custom

Because in all the config pages, there are much hiérarchic similarities, we decided to make one global config-UIMap, in which we put controls that can be found with similar properties on every config-page. For page-specific properties, we will make a separate UIMap.

*note that this section is not yet fully analysed and finished*

### 6.1 Dropdown grid

All Config-pages have 2 dropdown-buttons in the titlebar. Both of them open up a dropdown-window.

The first dropdown shows a list of different cases of the pages context (ex StageConfig --> different stages, PathConfig - different paths,...)

The second dropdown-window shows a list of versions we can use.

### 6.1.1 Click popup-window item

What?

Click on one of the items inside the popup-window and assert the correct action happened after you clicked.

How?

1. Add the listItem to the UI Map. Note that it is placed under the wrong hierarchy (wrong list).
2. To make sure you can find the control afterwards, add the SearchProperty "ClassName" to it, with the correct class-name.
3. Click the control
4. perform a read function to assert the correct elements are changed in the UI

## 6.2 Custom control bar

### 6.2.1 Change workbook state

Open overlay

What?

You can work in a workbook, and make as many changes as you want. After that you can lock the changes you made, validate them and publish them. All three functions are done with an overlay

how?

Refer to: States - Overlay state - Open overlay button

## 6.3 OptionsListView

In every config-page, there is an options list on the left of the screen. From there on, you can either choose the Info option or all the other options. The info displays info about this particular config-page. The rest of the options shows a list of other elements, which either modify data or contain objects of data you can modify.

## Testing Maät

In this section, we will describe the workflow of the testing of Maät, this includes a general method and a specific description of the process we went through for each page.

***Note that +-40% of all WorkbookConfig-tests are written, all other tests (ClinicHub, ...) are outdated and should be looked at to make them up to date again.***

### General work method

Here we will describe the general method to follow to make sure a page is fully tested.

It consists of 4 steps:

1. Analyse
2. Expand checklist
3. Map
4. Test

#### Analyse/expand checklist/map

These 3 steps have a lot of influence on each other. To analyse means that you manually search for all possible functionalities on a page by trying out all possibly usable controls in a number of different ways (ex. clicking, tabbing to the control, hovering over it, watching what happens when it's used,...). Simultaneously, you map all these usable controls to the checklist and add a "To do" to the test that has to be performed on this control. When you find that this functionality is not yet documented in the checklist, you add it to the checklist and analyse how you can test it (this is the "Expand checklist" step. Then you map all used controls to the UIMap with the UIMap test builder. How to do this is explained in the "How to test"-chapter. Sometimes however, the controls are badly mapped by the UIMap test builder. When this happens, you can try some of the following techniques:

1. Map the parent control and search for your desired control by using the "GetChildFromList"-function
2. Add a UID to the control in the XAML file (how to do this is explained in "How to test")
3. Change the searchproperties of the mapped control (for example instance, name,...) in such a way that when searching the control, the program will logically end up on the control you desire. This technique can also be used inside a test to customize your tests. For example, if there is a control on several sections of a page that looks visually the same, but is logically different in each section, you can write one tests in which you change the SearchProperty according to the different sections.

In the map-step, you also have to write a BaseClass for every page, in which you generate the controls you are about to use. This maintains the readability of every test project, because you generate each control only once. An example of this:

```

BaseClassWorkbookConfig.cs # <--> Caldos.Mast.CodedUITests
  namespace Caldos.Mast.CodedUITests.Screens.Config.WorkbookConfigPage.BaseClassW
    public class BaseClassWorkbookConfig : BaseClassConfigPages
    {
        #region Variables
        public XElement LockOverlayButton;
        public XElement OverlayCloseButton;
        public XElement OverlayCloseCloseButton;
        public XElement ClipboardButton;
        public XElement ClipboardMenu;
        public XElement CopyButton;
        public XElement ReadButton;
        public XElement Overlay;
        public XElement InfoBoxCloseButton;
        public XElement OverlayCloseCloseButton;
        public XElement BackButton;
        public XElement TitleText;
        public XElement StageConfigListItems;
        public XElement ContactViewRefListView;
        public XElement PathsViewRefListView;
        public XElement WorklistViewRefListView;
        public XElement VersionText;
        public XElement VskfButton;
        public UDMapWorkbookConfig.UICalidosMainWindow CopyworkbookWindow;
    }

BaseClassWorkbookConfig.cs # <--> Caldos.Mast.CodedUITests
  public void GenerateVariables()
  {
      GenerateConfigVariables();
      LockOverlayButton = UDMapWorkbookConfig.UICalidosMainWindow.UICurrentBarMainActionCustom.UIVergrendelenButton;
      InfoBoxCloseButton = UDMapWorkbookConfig.UICalidosMainWindow.UIDverlly.UITInfoBoxCloseCloseButton;
      OverlayCloseButton = UDMapWorkbookConfig.UICalidosMainWindow.UIDverlly.UITcloseButton;
      OverlayCloseCloseButton = UDMapWorkbookConfig.UICalidosMainWindow.UIDverlly.UITcloseCloseButton;
      UIVerboseButtons = UDMapWorkbookConfig.UICalidosMainWindow.UICurrentBarMainActionCustom.UIValiderenButton;
      PublishButton = UDMapWorkbookConfig.UICalidosMainWindow.UICurrentBarMainActionCustom.UIPublicerenButton;
      Overlay = UDMapWorkbookConfig.UICalidosMainWindow.UIDverlly;
      BackButton = UDMapWorkbookConfig.UICalidosMainWindow.UITbackButton;
      TitleText = UDMapWorkbookConfig.UICalidosMainWindow.UITtitleText;
      StageConfigListItems = UDMapWorkbookConfig.UICalidosMainWindow.UITitemList.UITstageConfigListItems;
      ContactViewRefListView = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UIContactViewRefListView;
      PathsViewRefListView = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UIPathsViewRefListView;
      WorklistViewRefListView = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UITworklistViewRefListView;
      WorkstepListItems = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UITwerkstapOpenText.UITitemList.UITitemListItems;
      UseThisVersionButton = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UITreversingbrukerButton;
      VersionText = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UITversionList.UITvsText;
      VskfButton = UDMapWorkbookConfig.UICalidosMainWindow.UITvskfSection.UITvskfButton;
      PopUpTitleEdit = UDMapWorkbookConfig.UICalidosMainWindow.UITpopUp.UITpopUpTitleEdit;
      PopUpCodeEdit = UDMapWorkbookConfig.UICalidosMainWindow.UITpopUp.UITcodeEdit;
      PopUpAddressButton = UDMapWorkbookConfig.UICalidosMainWindow.UITpopUp.UITvoegenenButton;
      CrtTrueOffTextTextBox = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UICrtTrueOffsetTextTextBox;
      ReferentieText = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UIRefertieText;
      HudlabelText = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UIHudlabelText;
      ReferentieStadumText = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UIRefertiestadumText;
      ReferentieVleistext = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UIRefertievleistext;
      HudlabelVleistext = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UIRefertievleistext;
      HudlabelNalezenenText = UDMapWorkbookConfig.UICalidosMainWindow.UITitemSection.UINalezenenText;
      RereadButton = UDMapWorkbookConfig.UICalidosMainWindow.UICurrentBarMainActionCustom.UITpopUpWindow.UINalezenenButton;
      ClipboardButton = UDMapWorkbookConfig.UICalidosMainWindow.UICurrentBarMainActionCustom.UILemboorButton;
  }

```

In this project, all the variables we will use once or multiple times, are generated. In every test project we will execute the function "GenerateVariables()" in the initialize-function. Also every private function we use in test projects can be placed in this project.

The hierarchy is as following: *PageParadigm* (ex *ClinicHubStates*) inherits from *BaseClassPage* (inherits from *BaseClassCategory*, sometimes) inherits from *BaseClassCodedUI*.

## Test

When all the previous is done, you can start writing all the tests you just analysed. Note that sometimes, different tests can be embedded in one test, like selecting and using a control, or navigation forward and backward. This is why sometimes more than one square in the checklist will be checked after writing one test.

## ConfigPages

### General problems

#### UIMapping

For all the tests (every category), the biggest problem is finding the control's you want to test. Because the UIMap-builder is not yet fully functional with windows 10 UWP applications, and some control's contain uIDs while others don't, many controls are mapped in a hierarchy that does not resemble the real hierarchy of the application. For example, some controls are mapped as children of one control, while they are really siblings. Or some controls are mapped directly under the Window, while they are children of for example a popup-window.

### General work method

#### UIMapping

In the config-pages, many controls come back with different values. This is why we made a general UIMap for all the config-pages, in which we put the controls that are available in every config-page. To make them accessible in every page, we sometimes had to change the searchproperties (for example remove the uID or add a certain instance). When doing this, make sure to first test this in every page with the UIMap-builder (you can click "search control" to make the UIMap-builder re-highlight the control you added). We also sometimes changed the SearchProperty "uID" to a part of the ID of the control, and change the property to "contains" instead of "equals"

For all the controls that are unique to one or some (but not all) pages we made a UIMap for every page.

#### Methods

For certain tests that will come back in every config-page, we wrote a method in which you have to pass some variables to specify in which page the test is executed or which page-specific controls are involved. We tried to write these methods as autonomous as possible. For the config-pages, we have a separate BaseClassCodedUI-partial class.

#### TestInitialize

Because navigating to the page directly results in a page without data (invalid), we have written the TestInitialize-method in such a way that it navigates to the TrialHub, with trialID 19 (which is Monaleesa28 or the trial with the most complete set of data). This trial is ideal for testing. After navigating to this trial, we wrote to search for the button that navigates to WorkbookConfig and clicking it. Afterwards we make the test wait until the page is fully loaded because much functionality in the ConfigPages only works optimally after the ProgressBar has disappeared.

For every config-page, we will write an extra line in the testinitialize-function to navigate from WorkbookConfig to this config-page.

### WorkbookConfig

#### General problems

Note that the hierarchy of the controls inside the HubSection varies from section to section although the visual representation is the same.

#### General work method

In the different option-sections of WorkbookConfig alone, we also found some controls that are always available, with different values. So here, we also added them to the UI Map with custom searchproperties.

## How to test

In this section, we will describe how to write tests for every item on the checklist. This is the actual implementation of the tests. In here, we will also describe some guidelines in the test code to keep the code readable and easily adaptable.

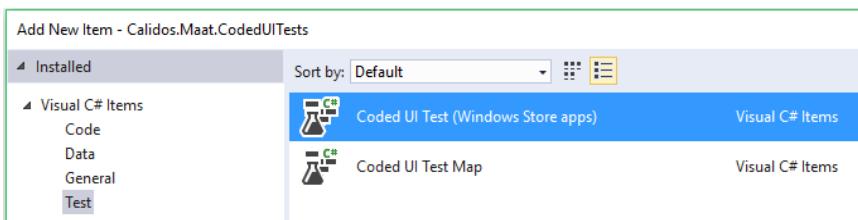
We will also discuss what steps we followed on each page to come to a correct test project.

### Basics

#### Coded UI Test Builder and UI Map

To add a Coded UI Test project:

- Right click on the map you want to add the project
  - Select 'Add'
  - Select 'New Item'
- Select 'Test'
  - Select 'Coded UI Test (Windows Store apps)'



To add a UI map:

- Do the same
- Select 'Coded UI Test Map' instead of 'Coded UI Test (Windows Store apps)'

When you added a Coded UI Test project, the first thing you need to do is add the right UI Map as a variable in your test project:

- At the top of your project, add a "using" statement for the UI Map you created. If we named our UI Map "UIMap\_ClinicHubPage", this is what we would have to add:

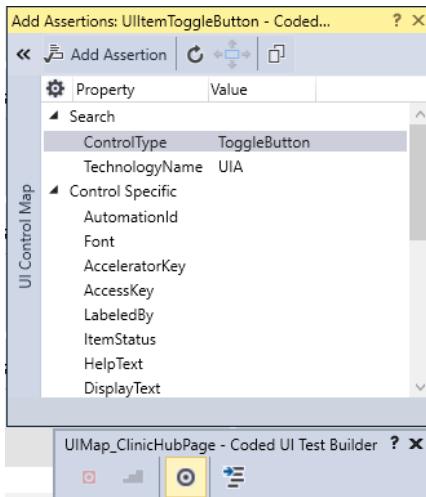
```
using Calidos.Maat.CodedUITests.Screens.Clinic.ClinicHubPage.UIMap_ClinicHubPageClasses;
```

- At the bottom of your project, change the UI Map property to something like this:

```
38 references | 2/2 passing | Tom Wuyts, 5 days ago | 1 author, 1 change
public UIMap_ClinicHubPage UIMapClinicHub
{
    get
    {
        if (map == null)
        {
            map = new UIMap_ClinicHubPage();
        }
        return map;
    }
}
private UIMap_ClinicHubPage map;
```

After doing this you can start writing tests. To write a test you first have to add the controls you want to use to the UI Map you created. To do this, right click on your '.uitest' file and select "Edit with Coded UI Test Builder".

The Coded UI Test Builder will launch itself. Don't worry if it minimizes your Visual Studio, the Builder just wants to indicate that you can start up your application you want to be mapping. By dragging the circular marker ('Add Assertions') onto the control you want to map you can add it to the UI Map. When you release the marker it will open another window, seen below.



The control you selected will be highlighted within the application with a blue border and on the right side of the new window you can see more detailed information about the currently selected control. If you click the arrow button in the top left hand corner you can see the hierarchy in which the control is embedded as well.

Note:

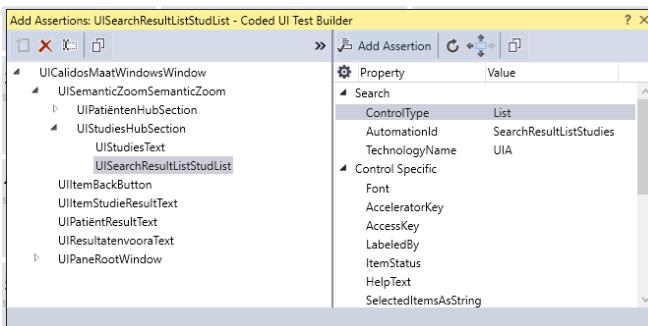
Because the Clinical Trials application 'Maät' is created as a Windows 10 Metro App, the Coded UI technology, mainly the Coded UI Test Builder, is not yet fully adapted for optimal hierarchy detection. To properly recognize the correct hierarchy, every control has to have a unique AutomationId. However, almost no control in the application has this Id. For example, to add ListItems we needed to figure out special techniques and workarounds in the test methods, which we will discuss later.

To make sure control's or lists are properly mapped and easy to find by the test program itself, we sometimes gave AutomationIDs to the control's ourselves. To do this, open the XAML file of the page you want to test, and then search for an indication out of which you can derive this section is the section you want to give an AutomationId.

Example: We gave a unique Id to a list, so we can later easily access the childs of that list

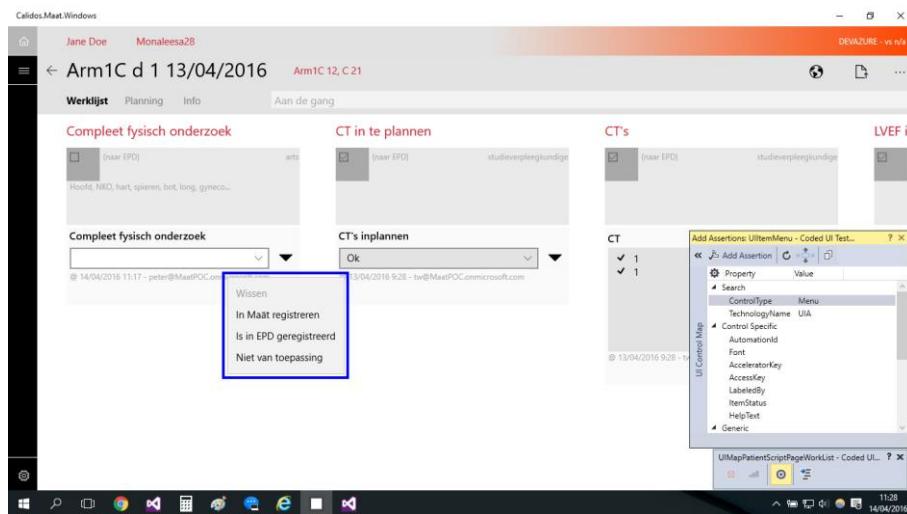
```
<GridViewItem>Studies</GridViewItem>
<GridViewItem>Patienten</GridViewItem>
</GridView>
<SemanticZoom.ZoomedOutView>
<SemanticZoom.ZoomedInView>
<Hub Style="{StaticResource ContentHubStyle}">
    <interactivity:Interaction.Behaviors>
        <semantic:HubSemanticZoomProviderBehavior GroupsLink="{StaticResource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

We wanted to give a unique Id to the list of Studies-search results, so we searched the XAML file for a while, tried naming some different grids and grid views, until we named the right one. Now if we select that list with the Coded UI Test Builder, the name we gave to it will appear as AutomationId, as seen below.



## Adding popup controls to the UI Map

Sometimes, we have to add control's or menu's to the UI Map that are only visible after clicking a certain button/control. If we just drag the crosshair of the CodedUITestBuilder onto the screen, the popup menu will disappear. A solution for this is using the "Ctrl-i"-combination while hovering over the menu, which will directly select the menu.



## Test Methods

### Test Classes

When you create a Coded UI Test class, the class must be marked as being for testing purposes. This is done by Visual Studio itself by adding the "[CodedUITest]" tag in front of the namespace declaration. More specific, for the application we are working on at the moment, is the "[CodedUITest(CodedUITestType.WindowsStore)]" tag that was auto-generated for us.

### The '[TestMethod]' tag

Every Coded UI Test needs to be preceded by a "[TestMethod]" tag, else the tests will not be picked up by Visual Studio's test framework.

A basic testmethod skeleton is depicted below.

```
/* Basic TestMethod skeleton:  
  
[TestMethod]  
public void METHODNAME()  
{  
}  
*/
```

#### The '[TestInitialize]' tag

This method will be executed **before every** '[TestMethod]'. It's important to note the 'every' in bold. Whenever a test method is ran, The testing framework loops over the whole test project to check for initialization methods. This method will always be ran first. Which could make it a useful method at certain points.

In our project, we will use this tag to navigate to the desired page in the application before every Test Method using Process() and a URI string that every page in the application possesses

#### The '[TestCleanup]' tag

This will be executed **after every** '[TestMethod]'. The same as with the TestInitialize, only after each and every test method. Usually this method contains a shutdown or close command to close the application after every test.

In our project this tag has not (yet) been used, because we don't want to reload all the data every few seconds for the UI tests.

#### The '[DataSource()]-tag

When you want to create a data driven test (which is a test using external data for execution/assertion/...), this tag can be added on top of the test method, together with the [TestMethod]-tag.

```
ex. [DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
 @"C:\dev\CTO\Src\Dev\Peter.0\Client\Calidos.Maat\Calidos.Maat.CodedUITests\Screens\Clinic\ClinicTrailsPage\NieuweStudieToevoegenData.csv", "NieuweStudieToevoegenData#csv",
 DataAccessMethod.Sequential), TestMethod]
```

Datasource uses 4 variables to connect to a datasource:

- type of data connection (ex. CSV, XML,...)
- path to datafile
- table name
- access method (sequential/random)

If you add a datatable to a test method, the method will be executed for every row in the table.

The first row will be ignored and can be used to assign variable names to your data for easy accessibility.

#### Testcontext

"TestContext", which is a property declared at the bottom of each test project, contains information about each test.

#### Data

when the test contains a datasource, data can be read through TestContext. Do do this, you can use the property "TestContext.DataRow[row]" in which row can be an index number or a title of the row you gave in your datatable (in string format)

## TestName & different TestInitialize-tags

In some test project, you will want to be able to use different TestInitialize methods. However the TestInitialize tag can only be used once. To solve this problem, you can write private functions with the different initialization methods, and in the TestInitialize method you can write an if-statement using the TestContext.TestName property. This is a string representation of the name you gave to your testmethod.

full example:

```
private void init(string searchWord)
{
    InitPlaybackSettings();

    _appStartUp.StartInfo.FileName = @"maat://Maat/Screens/Clinic/ClinicSearch?searchtext=" +
    searchWord;
    _appStartUp.Start();

    BackButton = new XamlControl(UIMapClinicSearch.UICalidosMaatWindowsWindow.UIItemBackButton);
}

private void initState1()
{
    init(SearchWord1);
}

private void initState2()
{
    init(SearchWord2);
}

private void initState3()
{
    init(SearchWord3);
}

[TestInitialize]
public void Initialize()
{
    if (TestContext.TestName.Contains("State1"))
    {
        this.initState1();
    }
    else if (TestContext.TestName.Contains("State2"))
    {
        this.initState2();
    }
    else if (TestContext.TestName.Contains("State3"))
    {
        this.initState3();
    }
}
```

## Generating controls

There are a few phases in order to generate a control in your test method:

- Identify the type of control you want to generate (XamlButton, XamlEdit,...).
  - XamlControl is a general name you can use for every type of Xaml control. The downside of doing so, is that you might not be able to use some control-specific properties.
- After you identified the control, give it a unique, yet obvious name.
- Finally, assign a path to your control referring to the control you added to the UI Map.
- After generating a control, you can use its properties to create different kinds of tests.

Some example of control properties often used in tests:

- Name      The name of the control
- Font      Text Font (handy when asserting if a tab is selected or not (Bold text))
- Exists      Bool that indicates if the control exist on the current page or not
- ...

Sometimes you want to use control-specific properties (like for example with ToggleButtons,...). Then you have to give a specific Xaml-type to the control (ex. XamlText, XamlEdit,...)

You will end up getting something like below image.

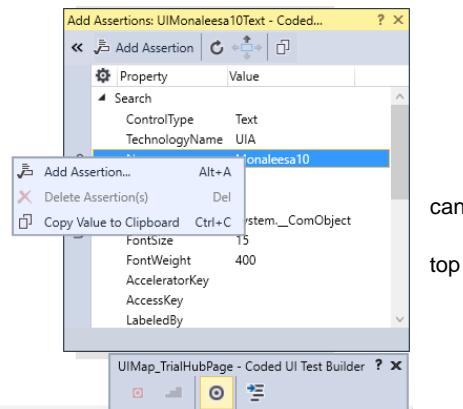
```
XamlText homeText = UIMapClinicHub.UICa
```

## Writing assertions

### (Semi) Automatic asserts

The Coded UI Test Builder enables you to write semi-automatic asserts.

After selecting a control with the marker, you make an assertion for a specific property, by selecting it and clicking "Add Assertion" in the left corner of the window, or by simply right clicking on the property you wish to assert and choosing "Add Assertion" as illustrated in the snapshot on the right.



A new window will pop up. Here you can choose the comparator, comparison value and message you will receive when the assertion fails.

In the example we go for the 'AreEqual' comparator, but there are some more things you could choose from, like 'Contains' or 'StartsWith'.

The comparison value is the value you want the actual value to be, in order to let the assertion pass. In other words, the test framework will get the actual value and check it against the comparison value

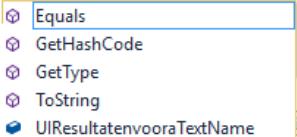
Additionally you can add a Message on assertion failure to get a better understanding what went wrong if an assertion failed. When done, click the 'OK' button.

Click the "Generate" button at the Coded UI Test Builder and your assertion a name. You can also add a description to make sure everyone knows what the assertion is all about. Your assertion is now ready. In order to use it, you have to call your UI Map first. Remember that everything that is generated with the Coded UI Test Builder is added to this map. After calling the UI Map you will see that your assertion is in the list of suggested code

```
UIMapClinicSearch.AssertBackButtonNavigation();
```

The UI Map in the example is the UI Map of the test project, and the highlighted area is the name you gave to your assertion method. The assertion contains the expected value you assigned on creation. However, on certain moments we will want to have a different expected value. The value can still be changed within your test methods or within the partial UI Map class (The part that doesn't contain auto-generated code). Changing the expected value can be done like below:

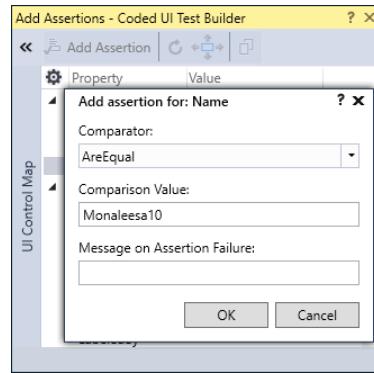
```
UIMapClinicSearch.AssertBackButtonNavigationExpectedValues.Equals
```



#### Manual assertions

The advantage of manual asserts is that you have much more control over the control you want to assert to as well as the expected values to make the test as autonomous as possible.

A manual assertion always starts with the 'Assert' keyword followed by the comparator. Note that there are different comparators available when making a manual assertion ('Contains' for example is not available when using manual assertions). The comparator requires parameters to create the full assertion.



```
Assert.AreEqual(true, hub.TryFind());
```

In the example, where we make an 'AreEqual' assertion, the first parameter is the expected value. The second parameter is the actual value. Where these 2 values come from, is completely under your control. This way of writing assertions is way more intensive than the auto-assertions, but makes room for some more flexibility. Manual assertions are written inside your test methods.

#### Assert.Fail()

This is a function, when executed, which automatically makes the test fail. You can give a string to the function in which you describe the error occurring if the function is executed. Assert.Fail() is an easy way of making a test fail when a certain condition is true (for example a control is visible when it should not be visible). In other words, for some tests we will write "if" statements in which faulty conditions are specified. Inside these "if" statements we will write an Assert.Fail().

ex.

```
public void ClinicTrialsPageOverlayClickCloseOverlayButton()
{
    XamlControl OverlayHub = UIMapClinicTrials.UICalidosMaatWindowsWindow.UIEennieuwestudietoevoHub;
    XamlControl CloseButton = UIMapClinicTrials.UICalidosMaatWindowsWindow.UICloseOverlayButtonButton;
    OverlayHub.TryFind();
    UseControl(CloseButton, Action.Click);
    Playback.Wait(delayMilliseconds);
    if (OverlayHub.Left > 0) Assert.Fail();
}
```

#### Commonly used variables and methods

These are some variables and methods that already exist, and that we will use to write some of our tests.

#### StopWatch

This is an object with which you can time how long it takes for a certain amount of code to execute. To use it, add "System.Diagnostics" to your usings. You can then generate a new StopWatch. To start timing, write StopWatch.Start(), and to read how many time has passed, use the StopWatch.ElapsedMilliseconds-property. Write your code between the StopWatch.Start()-function and StopWatch.Stop()-function.

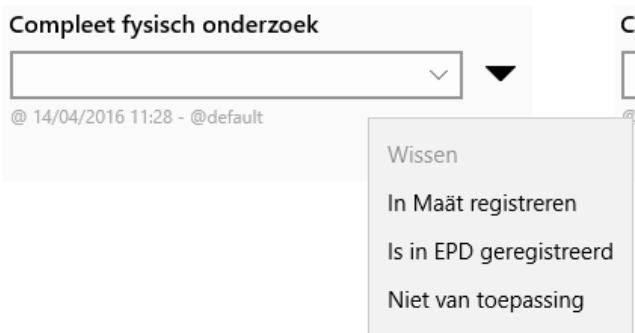
#### Commonly used controls

These are some control's that we found in the application, for which you have to use certain techniques to use them.

#### Flyout button

These are buttons that make a popup-menu appear when clicking on it. In this popup-menu, you can choose several options, each resulting in a different change somewhere else in the application.

ex.



The thick black arrow is a flyout button. Clicking one of the options in the popup-menu, has an influence on the combobox below "compleet fysisch onderzoek".

To add the menu appearing after clicking the flyout button to the UI Map, you can't drag the crosshair of the UI Map-builder to the menu, because clicking elsewhere makes the menu disappear. Instead, hover over the menu with your mouse and use the "Ctrl-i" combination.

#### ProgressBar

Sometimes, a control is already loaded and enabled before a page is fully loaded. However, the desired functionality of the control will only work correctly after the page is fully loaded. In that case, it's necessary to build some kind of waiting function to wait for the page to be fully loaded.

This is where the ProgressBar comes in handy. It's a bar on top of the screen, which displays an animation as long as the page is loading. As soon as the page is loaded, the bar disappears.

To setup a waiting function for the page to be fully loaded, you can add the ProgressBar to the UI Map with the "Ctrl-l" key combination (do this while a page is still loading, before it disappears). Afterwards, you can use one of the coördinates of the ProgressBar to build an empty while loop. The coördinates of a control that has disappeared are set to "-1", so your while-loop will look something like this:

```
"while (ProgressBar.Top > 0);"
```

Note that no code is executed in the while-loop, it is just there to wait for the condition you desire to occur.

## BaseClassCodedUI

This is a base class we wrote ourselves, in which we write functions for actions we have to use very often. Like this, we can very easily perform the action multiple times in just one line of code. We will also add commonly used variables and global test scenarios (that don't require page-specific control's but overall control's, available on every page) to this class.

Every test project that is created, has to inherit from this base class.

### General

#### enum Action

This is an enum of actions that we can use for all methods that involve performing some sort of action on a control. This enhances the readability of the method.

```
public enum Action
{
    Enter,
    Space,
    Click,
    CtrlKey,
    CtrlScroll,
    Tab,
    Arrows
};
```

#### enum Scroll

Scroll is an enum to indicate whether you want to scroll up or down, this enum is also written to enhance the readability of the methods that use scroll functions.

```
public enum Scroll
{
    Up,
    Down
};
```

#### DelayMilliseconds

To write a hardcoded delay in a test, we use the PlayBack.Wait() function, in which you have to pass an int indicating the amount of milliseconds the delay has to be. We created this variable to pass to the delay's so we can edit the wait-time of every delay in the entire project in one line.

```
protected int DelayMilliseconds = 2000;
```

#### Process AppStartUp

AppStartUp is a variable used to start the application before every test. AppStartUp has a property called StartInfo, in which you can pass the Url of the page you want to navigate to. Afterwards, you can use the function Start() of AppStartUp to start the application on the desired page.

```
protected Process AppStartUp = new Process();
```

### InitMethodSettings

This is a method we use in every TestInitialize-function, to set some parameters regarding the execution of the test.

- DelayBetweenActions: the time the test normally waits between every action it has to execute (ex. between 2 clicks)
- SearchTimeout: the maximum time the tests dedicates on searching a control on the UI, if exceeded, the test fails and the error indicates the control cannot be found
- WaitForReadyTimeout: the maximum time the test dedicates on the total execution of the test, if exceeded, the test will fail no matter what.
- ThinkTimeMultiplier: This multiplies all other delay-values of the playbacksettings to its value

```
protected void InitMethodSettings()
{
    Playback.PlaybackSettings.DelayBetweenActions = 200; //200ms
    Playback.PlaybackSettings.SearchTimeout = 20000; //20s
    Playback.PlaybackSettings.WaitForReadyTimeout = 30000; //30s
    Playback.PlaybackSettings.ThinkTimeMultiplier = 2;
}
```

### RegisterTest

```
1 reference | Tom Wuyts | 1 author, 1 change
public abstract class BaseTestClass
{
    485 references | 0/232 passing | Tom Wuyts | 1 author, 1 change
    public void RegisterTest(string testCategory, string testObject)
    {
        Trace.WriteLine(string.Format("MaatTest|{0};{1};", testCategory, testObject));
    }
}
```

For the explanation of this function, refer to "Automation of Coded UI Tests and its results".

### GetChildFromList

This is a function written to get one specific child from a list or control. We found that sometimes a list or control is already loaded before its children are loaded, so we had to build a manual delay that can detect when the children of a list or control are loaded. We then added a variable to already select the instance you want to get.

- list: the list inside which the control you want to get exists
- instance: the instance of the control you want to get inside the list

The function returns the control you asked for.

```

public XamlControl GetChildFromList(XamlControl list, int instance)
{
    list.WaitForControlExist();
    UITestControlCollection listItems = list.GetChildren();
    while (listItems.Count == 0)
    {
        listItems = list.GetChildren();
        Thread.Sleep(100);
    }
    XamlControl testCase = listItems[instance] as XamlControl;
    return testCase;
}

```

### WaitForPageLoaded

This is a function that uses the ProgressBar in a delay to wait for the page to be fully loaded.  
When the progressbar disappears, the page is fully loaded.

```

public void WaitForPageLoaded()
{
    if (UIMapGeneral.UICalidosMaatWindowsWindow.UIProgressBar.Exists)
        while (UIMapGeneral.UICalidosMaatWindowsWindow.UIProgressBar.Top > 0) Thread.Sleep(100) ;
}

```

### ChangeNameSearchProperty

```

/// <summary>
/// Changes the searchproperty "name" to the desired value
/// </summary>
/// <param name="_control">control to change the searchproperty from</param>
/// <param name="_uID">Sesired "name"-searchproperty</param>
5 references | 0/5 passing | Peter Van de Putte, 8 days ago | 1 author, 1 change
public void ChangeNameSearchProperty(XamlControl _control, string _uID)
{
    _control.SearchProperties.Remove("Name");
    _control.SearchProperties.Add("Name", _uID);
}

```

### TabToControl

This is a function that keeps tabbing until the control that is passed to it is highlighted.

```

public void TabToControl(XamlControl control)
{
    while (control.HasFocus == false)
    {
        Keyboard.SendKeys("{TAB}");
        Thread.Sleep(100);
    }
}

```

### GoToControl

This function is used for a control in a list. It first gets the first control in the list in which the control you want to highlight exists. It then uses the “Up”-arrow key until the first control in the list is highlighted, and then it uses the “Down”-arrow key until the control you want to highlight is highlighted.

```

public void GoToControl(XamlControl control)
{
    UITestControl parent = control.GetParent();
    UITestControlCollection childs = parent.GetChildren();
    while (childs[0].HasFocus == false)
    {
        if (control.HasFocus == true)
            return;
        else
            Keyboard.SendKeys("{UP}");
        Thread.Sleep(100);
    }
    while (control.HasFocus == false)
    {
        Keyboard.SendKeys("{DOWN}");
        Thread.Sleep(100);
    }
}

```

#### ScrollIToControl

This function accepts 3 parameters. The control to which it has to scroll, a "scrollControl", which is a control from which it has to start scrolling, and a "Scroll"-enum, which can be Up or Down.

The function makes the mouse hover over the "scrollControl", and then starts scrolling in the direction of "scroll", until "control" is visible.

```

public void ScrollIToControl(XamlControl control, XamlControl scrollControl, Scroll scroll)
{
    Mouse.Hover(GetMiddleOfControl(scrollControl));
    while (control.Left < 0)
    {
        if (scroll == Scroll.Up)
            Mouse.MoveScrollWheel(1);
        else
            Mouse.MoveScrollWheel(-1);
        Thread.Sleep(100);
    }
}

```

#### WriteAsUser

This function converts the string "msg" to char's and then inputs every char as if a user was typing the keyboard.

```

public void WriteAsUser(string msg)
{
    char[] msgInChars = msg.ToCharArray();
    for (int i = 0; i < msgInChars.Length; i++)
    {
        Keyboard.SendKeys(msgInChars[i].ToString());
    }
}

```

### GetMiddleOfControl

This function returns a Point, which is the middle of the "control" passed to it.

```
public Point GetMiddleOfControl(XamlControl control)
{
    int x = control.Left + (control.Width / 2);
    int y = control.Top + (control.Height / 2);

    Point middle = new Point(x, y);

    return middle;
}
```

### GetExpectedValue

This function returns a string, which is the expected value to assert to in certain tests. You give 2 parameters to the function, the first is the control in which the value you expect exists, and the second is the instance of the value inside the control.

```
public string GetExpectedValue(XamlControl testCase, int instance)
{
    string expectedValue = GetChildFromList(testCase, instance).Name;
    return expectedValue;
}
```

### RandomListInstance

This function accepts a list, and then returns a random int between 1 and the lists count -2 (so a random instance which is not the first and not the last instance of the list).

```
public int RandomListInstance(XamlControl list)
{
    UITestControlCollection listItems = list.GetChildren();
    while (listItems.Count == 0)
    {
        listItems = list.GetChildren();
        Thread.Sleep(100);
    }

    Random rdm = new Random();
    int random = rdm.Next(1, listItems.Count - 2);
    return random;
}
```

### LastListInstance

The same as above, but this time the function returns the last instance inside the list as an int.

```
public int LastListInstance(XamlControl list)
{
    UITestControlCollection listItems = list.GetChildren();
    while (listItems.Count == 0)
    {
        listItems = list.GetChildren();
        Thread.Sleep(100);
    }

    int last = listItems.Count - 1;
    return last;
}
```

## Content

### ReadListItems

This function reads all the content of all the items inside a list, and compares it to an expected value string. If the string is present somewhere inside the list, the test passes, if the string is not in the list, the test fails.

```
public void ReadListItems(XamlControl list, string expectedResult)
{
    XamlControl tile = GetChildFromList(list, 0);
    string text;
    for (int i = 0; i < list.GetChildren().Count; i++)
    {
        bool correct = false;
        tile = GetChildFromList(list, i);
        for (int j = 0; j < tile.GetChildren().Count; j++)
        {
            text = GetChildFromList(tile, j).Name;
            if (text.ToLower().Contains(expectedResult.ToLower())) correct = true;
        }
        if (correct == false) Assert.Fail("The search algorithm went wrong");
    }
}
```

## Navigations

### NavigateVariableControl

This is a function which tests the navigation for variable control's. Variable controls are controls that exist inside a list but are dependent on the data existing inside the database. The amount of control's inside the list is not fixed and neither is the data in the controls.

The function accepts a high amount of variables necessary for the testing:

- **list**: List inside which the case you want to test exists
- **assertControl**: Control you want to assert to after forward navigation
- **scrollStartControl**: Control from which you want to start the scroll function
- **backButton**: Button to navigate backwards after forward navigation
- **pageTitle**: Title of the page from which you are navigating
- **expectedValue**: Value you expect in the title after navigating forwards
- **instance**: instance of the ListItem you want to use inside the list
- **action**: Execution method (Click/Enter/Space)

```

public void NavigateVariableControl(XamlControl scrollStartControl, XamlControl scrollEndControl, Action forwardAction, Action backwardAction, string pageTitleControlName, string assertControlName, string expectedValue)
{
    string pageTitle = pageTitleControlName;
    XamlControl testCase = GetChildFromList(list, instance);
    if (forwardAction == Action.Click)
    {
        ScrollToControl(testCase, scrollStartControl, Scroll.Down);
        UseControl(testCase, Action.Click);
    }
    else
    {
        ListItemKey(list, instance, forwardAction);
    }

    string value = assertControlName;
    while (value.Contains(pageTitle) || value.Equals(""))
    {
        value = assertControlName;
        Thread.Sleep(100);
    }
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(expectedValue, value);

    UseControl(backButton, backwardAction);

    string backValue = assertControlName;
    while (backValue.Contains(value) || backValue.Equals(""))
    {
        backValue = assertControlName;
        Thread.Sleep(100);
    }
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(pageTitle, backValue);
}

```

#### NavigateHyperLink

This function tests the navigation of a hyperlink-control. The parameters you give to it are the hyperlink, the title-control of the pages and the action you want to use (click/enter/space).

```

public void NavigateHyperlink(XamlHyperlink _hyperLink, XamlControl _titleControl, Action _action)
{
    XamlControl _backButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIItemBackButton;
    string _startValue = _titleControl.Name;
    string _trialLinkName = _hyperLink.Name;
    UseControl(_hyperLink, _action);
    while (_titleControl.Name.Contains(_startValue) || _titleControl.Name.Equals("")) ;

    if (_trialLinkName != _titleControl.Name) Assert.Fail("The forward navigation went wrong!");

    UseControl(_backButton, _action);

    while (_titleControl.Name.Contains(_trialLinkName) || _titleControl.Name.Equals("")) ;
    if (_startValue != _titleControl.Name) Assert.Fail("The backward navigation went wrong!");
}

```

## NavigateFixedControlToPage

This function tests the navigation of a fixed control to a unique page. Fixed controls are controls that always exist on a page and in this case, each control navigates to a unique page in relation to the other controls in the same collection.

These are the variables you need for the function:

- **hub**: Hub in which the ListItem you want to click exists
- **backButton**: Backbutton to navigate back to the page you are testing
- **assertControl**: The titlecontrol of the page you are navigating to
- **pageTitleControl**: The titlecontrol of the page from which you are navigating
- **scrollStartControl**: Control from which you want to start scrolling
- **expectedValue**: Value to expect in the title after navigating forwards
- **instance**: Instance of the button you want to click inside the hub
- **action**: Execution method (click/enter/space)

```
public void NavigateFixedControlToPage(XamlControl scrollStartControl, XamlControl backButton, XamlControl hub, XamlControl instance)
{
    XamlControl testButton = GetChildFromList(hub, instance);
    string pageTitle = pageTitleControl.Name;

    if (action == Action.Click) ScrollToControl(testButton, GetChildFromList(scrollStartControl, 0), Scroll.Down);
    UseControl(testButton, action);

    string value = assertControl.Name;
    while (value.Contains(pageTitle) || value.Equals(""))
    {
        value = assertControl.Name;
        Thread.Sleep(100);
    }
    value = assertControl.Name;
    Assert.AreEqual(expectedValue, value);

    UseControl(backButton, action);

    string backValue = pageTitleControl.Name;
    while (backValue.Contains(value) || backValue.Equals(""))
    {
        backValue = pageTitleControl.Name;
        Thread.Sleep(100);
    }
    backValue = pageTitleControl.Name;
    Assert.AreEqual(pageTitle, backValue);
}
```

## NavigateFixedControlToTab

Performs the same test as `NavigateFixedControlToPage`, but the difference is that these controls, in their own collection, all navigate to the same page but trigger a different state of that page. In this case, they all trigger a different tab to open on the same page.

Variables:

- **hub**: Hub in which the Control you want to click exists
- **backButton**: Backbutton to navigate back to the page you are testing
- **assertTabs**: The tab-bar in which you can go to the different tabs, on the page you are navigating to
- **scrollStartControl**: Control from which you want to start scrolling
- **buttonInstance**: Instance of the button you want to click inside the hub
- **tabInstance**: Instance of the tab inside "assertTabs" the button should navigate to
- **action**: Execution method (click/enter/space)

```

public void NavigateFixedControlToTab(XamlControl scrollStartControl, XamlControl backButton, XamlControl hub, XamlControl testButton, Action action)
{
    XamlControl testButton = GetChildFromList(hub, buttonInstance);

    if (action == Action.Click) ScrollToControl(testButton, GetChildFromList(scrollStartControl, 0), Scroll.Down);
    UseControl(testButton, action);

    bool ready = false;
    int i = 0;
    XamlToggleButton tab = GetChildFromList(GetChildFromList(assertTabs, i), 0) as XamlToggleButton;
    while (ready == false)
    {
        if (tab.Pressed == true) ready = true;

        tab = GetChildFromList(GetChildFromList(assertTabs, i), 0) as XamlToggleButton;
        if ((i + 1) < assertTabs.Children().Count)
        {
            i++;
        }
        else i = 0;
        Thread.Sleep(100);
    }
    tab = GetChildFromList(GetChildFromList(assertTabs, tabInstance), 0) as XamlToggleButton;
    Assert.AreEqual(true, tab.Pressed);

    UseControl(backButton, action);
    Assert.AreEqual(true, hub.TryFind());
}

```

## States

### GetHubSections

This is a function that returns all the hubsections that exist inside a hub as a Testcontrolcollection.

```

public UITestControlCollection GetHubSections(XamlControl hub)
{
    hub.WaitForControlExist();
    UITestControlCollection hubSections = hub.GetChildren();
    int count = hubSections.Count;
    do
    {
        count = hubSections.Count;
        Playback.Wait(100);
        hubSections = hub.GetChildren();
        Thread.Sleep(100);
    } while (hubSections.Count == 0 || count < hubSections.Count);

    return hubSections;
}

```

### CheckSemanticZoomTitles

This function zooms out the semantic zoom and checks if the titles displayed in zoomed out view represent the same titles as displayed in the zoomed in view.

```

public void CheckSemanticZoomTitles()
{
    UITestControlCollection zoomedInHubSections = GetHubSections(UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom.UITemHub);

    ZoomOutSemanticZoom(Action.CtrlScroll, UIMapGeneral.UICalidosMaatWindowsWindow.USemanticZoomSemanticZoom);

    UITestControlCollection zoomedOutHubSections = GetHubSections(UIMapGeneral.UICalidosMaatWindowsWindow.USemanticZoomSemanticZoom.UITemList);

    if (zoomedInHubSections.Count != zoomedOutHubSections.Count) Assert.Fail("The zoomed out view of the Semantic Zoom doesn't have the same amount of items");
    for (int i = 0; i < zoomedInHubSections.Count; i++)
    {
        Assert.AreEqual(zoomedInHubSections[i].Name, GetChildFromList(zoomedOutHubSections[i] as XamlControl, 0).Name);
    }

    ZoomInSemanticZoom(zoomedOutHubSections[0] as XamlControl, Action.CtrlScroll, UIMapGeneral.UICalidosMaatWindowsWindow.USemanticZoomSemanticZoom);
}

```

## ZoomOutSemanticZoom

To go from zoomed-in to zoomed-out state, you can use this function. The action you give to the function is the action you want to zoom out with, this can be Ctrl+Scroll, Ctrl+key,...

```
public void ZoomOutSemanticZoom(Action action, XamlSemanticZoom semanticZoom)
{
    semanticZoom.WaitForControlExist();

    if (semanticZoom.ZoomedIn)
    {
        Playback.Wait(2000);
        if (action == Action.CtrlScroll)
        {
            Mouse.Hover(GetMiddleOfControl(semanticZoom));
            Keyboard.PressModifierKeys(ModifierKeys.Control);
            Mouse.MoveScrollWheel(-1);
            Keyboard.ReleaseModifierKeys(ModifierKeys.Control);
        }

        else if (action == Action.CtrlKey)
        {
            Mouse.Click(new Point(semanticZoom.Left, semanticZoom.Top));
            Keyboard.PressModifierKeys(ModifierKeys.Control);
            Keyboard.SendKeys("{SUBTRACT}");
            Keyboard.ReleaseModifierKeys(ModifierKeys.Control);
        }
    }

    if (semanticZoom.ZoomedIn) Assert.Fail("The Semantic zoom is still zoomed in after trying to zoom out!");
}
```

The function below is the same, but for the click-execution

```
public void ZoomOutSemanticZoom(XamlControl control, XamlSemanticZoom semanticZoom)
{
    semanticZoom.WaitForControlExist();

    if (semanticZoom.ZoomedIn)
    {
        Mouse.Click(GetMiddleOfControl(control));
        Assert.AreEqual(false, semanticZoom.ZoomedIn);
    }

    if (semanticZoom.ZoomedIn) Assert.Fail("The Semantic zoom is still zoomed in after trying to zoom out!");
}
```

## ZoomInSemanticZoom

This function does the same as the above function, except this one goes from the zoomed-out state back to the zoomed-in state instead of the other way around.

```
public void ZoomInSemanticZoom(XamlControl control, Action action, XamlSemanticZoom semanticZoom)
{
    semanticZoom.WaitForControlExist();

    if (!semanticZoom.ZoomedIn)
    {
        if (action == Action.CtrlScroll)
        {
            Mouse.Hover(GetMiddleOfControl(control));
            Keyboard.PressModifierKeys(ModifierKeys.Control);
            Mouse.MoveScrollWheel(1);
            Keyboard.ReleaseModifierKeys(ModifierKeys.Control);
        }

        else if (action == Action.CtrlKey)
        {
            GoToControl(control);
            Keyboard.PressModifierKeys(ModifierKeys.Control);
            Keyboard.SendKeys("{ADD}");
            Keyboard.ReleaseModifierKeys(ModifierKeys.Control);
        }

        else if (action == Action.Click)
        {
            Mouse.Click(GetMiddleOfControl(control));
        }
    }

    if (!semanticZoom.ZoomedIn) Assert.Fail("The Semantic zoom is still zoomed out after trying to zoom in!");
}
```

### AssertSemanticZoom

This function checks if the semantic zoom zoom functions work correctly. There are 2 overloads of this function. One overload for the key-executions and one for the click-execution.

Each ask for the hubsection and the hubsection-name control.

```
public void AssertSemanticZoom(XamlControl hubsection, XamlControl hubsectionName, Action action)
{
    XamlSemanticZoom semanticZoom = UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom;
    semanticZoom.WaitForControlExist();
    UITestControlCollection zoomedInHubSections;
    UITestControlCollection zoomedOutHubSections;

    if (semanticZoom.ZoomedIn) zoomedInHubSections = GetHubSections(UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom.UIItemList);
    else zoomedOutHubSections = GetHubSections(UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom.UIItemList);

    ZoomOutSemanticZoom(action, semanticZoom);

    ZoomInSemanticZoom(hubsectionName, action, semanticZoom);
    hubsection.WaitForControlExist();
    Assert.IsTrue(hubsection.Left >= semanticZoom.Left);
}

public void AssertSemanticZoom(XamlControl hubsection, XamlControl hubsectionName, XamlControl control)
{
    XamlSemanticZoom semanticZoom = UIMapGeneral.UICalidosMaatWindowsWindow.UISemanticZoomSemanticZoom;
    ZoomOutSemanticZoom(control, semanticZoom);
    ZoomInSemanticZoom(hubsectionName, Action.Click, semanticZoom);
    hubsection.WaitForControlExist();
    Assert.IsTrue(hubsection.Left >= semanticZoom.Left);
}
```

### Functionality

#### UseHomeButton

This function uses the HomeButton with the action you give to it. Afterwards, it checks if the correct action happened after using the button.

```
public void UseHomeButton(Action action)
{
    XamlControl homeButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIPaneRootWindow.UIToggleButton;
    XamlText homePageText = UIMapGeneral.UICalidosMaatWindowsWindow.UITitleText;

    homeButton.TryFind();
    bool initialState = (bool)homeButton.GetProperty("Pressed");

    UseControl(homeButton, action);
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(!initialState, homeButton.GetProperty("Pressed"));
    Assert.AreEqual(true, homePageText.GetProperty("Exists"));

    UseControl(homeButton, action);
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(initialState, homeButton.GetProperty("Pressed"));
}
```

### UseExpandMenuButton

This function tests the functionality of the “expand”-button, below the home-button. You can choose the action with which you want to execute the “expand” action.

```
public void UseExpandMenuButton(Action action)
{
    XamlControl expandButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIPaneRootWindow
        .UIExpandToggleButton;
    expandButton.WaitForControlExist();
    bool initialState = (bool)expandButton.GetProperty("Pressed");

    // Variable value, indicating the threshold width between expanded and collapsed menu - Adjust if necessary
    int threshold = 50;

    Playback.Wait(DelayMilliseconds);
    if (!initialState)
        Assert.IsTrue((int)expandButton.GetProperty("Width") > threshold);
    else
        Assert.IsTrue((int)expandButton.GetProperty("Width") < threshold);

    UseControl(expandButton, action);
    expandButton.WaitForControlExist();
    Assert.AreEqual(!initialState, expandButton.GetProperty("Pressed"));

    Playback.Wait(DelayMilliseconds);
    if (!initialState)
        Assert.IsTrue((int)expandButton.GetProperty("Width") > threshold);
    else
        Assert.IsTrue((int)expandButton.GetProperty("Width") < threshold);

    UseControl(expandButton, action);
    expandButton.WaitForControlExist();
    Assert.AreEqual(initialState, expandButton.GetProperty("Pressed"));

    Playback.Wait(DelayMilliseconds);
    if (initialState)
        Assert.IsTrue((int)expandButton.GetProperty("Width") > threshold);
    else
        Assert.IsTrue((int)expandButton.GetProperty("Width") < threshold);
}
```

### UseSettingsButton

This function tests the settings-button. You can choose the action to execute with, note that the navigation doesn't exist yet.

```
public void UseSettingsButton(Action action)
//TODO: Implement navigation when button actually does something
{
    XamlControl settingsButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIPaneRootWindow
        .UISettingsToggleButton;

    settingsButton.WaitForControlExist();
    bool initialState = (bool)settingsButton.GetProperty("Pressed");

    UseControl(settingsButton, action);
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(!initialState, settingsButton.GetProperty("Pressed"));
    //Check navigation here

    UseControl(settingsButton, action);
    Playback.Wait(DelayMilliseconds);
    Assert.AreEqual(initialState, settingsButton.GetProperty("Pressed"));
}
```

### UseControl

This is a function designed to use a control with the action given to it. If the action is "click", the control will be clicked. If the action is "enter", the control will be entered,...

The reason we wrote this function is to implement the "WaitForControlExist()" function so we do not have to write this line of code every time we want to use a control.

```
public void UseControl(XamlControl control, Action action)
{
    if (action == Action.Enter || action == Action.Space)
    {
        TabToControl(control);
        Keyboard.SendKeys("{" + action.ToString().ToUpper() + "}");
    }
    if (action == Action.Click)
    {
        control.WaitForControlExist();
        Mouse.Click(GetMiddleOfControl(control));
    }
}
```

### CheckReactionSpeed

This function returns a TimeSpan, which contains the amount of time it took for the page you are testing to fully load.

```
public TimeSpan CheckReactionSpeed()
{
    XamlProgressBar _progressBar = UIMapGeneral.UICalidosMaatWindowsWindow.UIProgressBar;
    Stopwatch _watch = new Stopwatch();
    _watch.Start();
    while (_progressBar.Top > 0) ;
    _watch.Stop();

    return _watch.Elapsed;
}
```

### ComboBoxSelectItem

This function is designed to select an item from a combobox. You give the item you want to select to it, and the function opens the combobox and clicks that item.

```
public void ComboBoxSelectItem(XamlControl control, int number)
{
    control.WaitForControlExist();
    Mouse.Click(GetMiddleOfControl(control.GetParent() as XamlControl));
    UseControl(control, Action.Click);
}
```

## ListItemKey

This is a method with which you can use a ListItem inside a list, using the keyboard. The parameters are:

- **list**: the list inside which your ListItem exists.
- **instance**: the instance of the your ListItem inside the list
- **action**: the execution method

```
public void ListItemKey(XamlControl list, int instance, Action action)
{
    XamlControl testCase = this.GetChildFromList(list, instance);

    if (instance == 0)
    {
        TabToControl(testCase);
    }

    else if (instance != 0)
    {
        while (list.GetChildren()[0].HasFocus == false)
        {
            Keyboard.SendKeys("{TAB}");
            Thread.Sleep(100);
        }
        GoToControl(testCase);
    }
    Keyboard.SendKeys("{" + action.ToString() + "}");
}
```

## ClearEdit

This method is designed to clear an edit-control of all it's content, before putting new content in it.

```
public void ClearEdit(XamlEdit edit)
{
    int i = 0;
    while ((edit.Text != "") && (i < 20))
    {
        i++;
        Keyboard.SendKeys("{Back}");
        Thread.Sleep(100);
    }
}
```

## UseSearchBox

```
/// <summary>
/// Write text as if typed by user
/// </summary>
1 reference | Peter Van de Putte | 1 author, 1 change
public void UseSearchbox(XamlControl searchbox, string msg, Action action)
{
    WriteAsUser(msg);
    if (action == Action.Click)
        Mouse.Click(new Point((searchbox.Left + searchbox.Width) - (searchbox.Height / 2), searchbox.Top + (searchbox.Height / 2)));
    if (action == Action.Enter)
        Keyboard.SendKeys("{" + action.ToString().ToUpper() + "}");
}
```

```
RandomClose
/// <summary>
/// Clicks on a coordinate outside of the control to close the control
/// </summary>
/// <param name="_control">the control you want to close</param>
4 references | 0/4 passing | Peter Van de Putte, 13 days ago | 1 author, 2 changes
public void RandomClose(XamlControl _control)
{
    Mouse.Click(new Point(_control.Left + _control.Top + 5, _control.Top + _control.Height + 5));
    Playback.Wait(2000);
}
```

## Config

For all the config-pages, we made a separate BaseClassCodedUI-partial class, because there are many tests that can be copy-pasted inside these pages.

## NavigateHyperlinkConfig

This is a function that tests the navigation through the hyperlink on top of every config page.

```
public void NavigateHyperlinkConfig(Action _action)
{
    XamlControl _backButton = UIMapGeneral.UICalidosMaatWindowsWindow.UIItemBackButton;

    string _startValue = UIMapConfig.UICalidosMaatWindowsWindow.UIPageTitleText.Name;
    string _trialLinkName = UIMapConfig.UICalidosMaatWindowsWindow.UITrialHyperlink.Name;
    UseControl(UIMapConfig.UICalidosMaatWindowsWindow.UITrialHyperlink, _action);
    while (UIMapConfig.UICalidosMaatWindowsWindow.UIPageTitleText.Exists) Thread.Sleep(100);
    string val = UIMapConfig.UICalidosMaatWindowsWindow.UINavigatedTitleText.Name;
    if (_trialLinkName != UIMapConfig.UICalidosMaatWindowsWindow.UINavigatedTitleText.Name) Assert.Fail("The forward navigation went wrong!");

    UseControl(_backButton, _action);
    WaitForPageLoaded();
    if (_startValue != UIMapConfig.UICalidosMaatWindowsWindow.UIPageTitleText.Name) Assert.Fail("The backward navigation went wrong!");
}
```

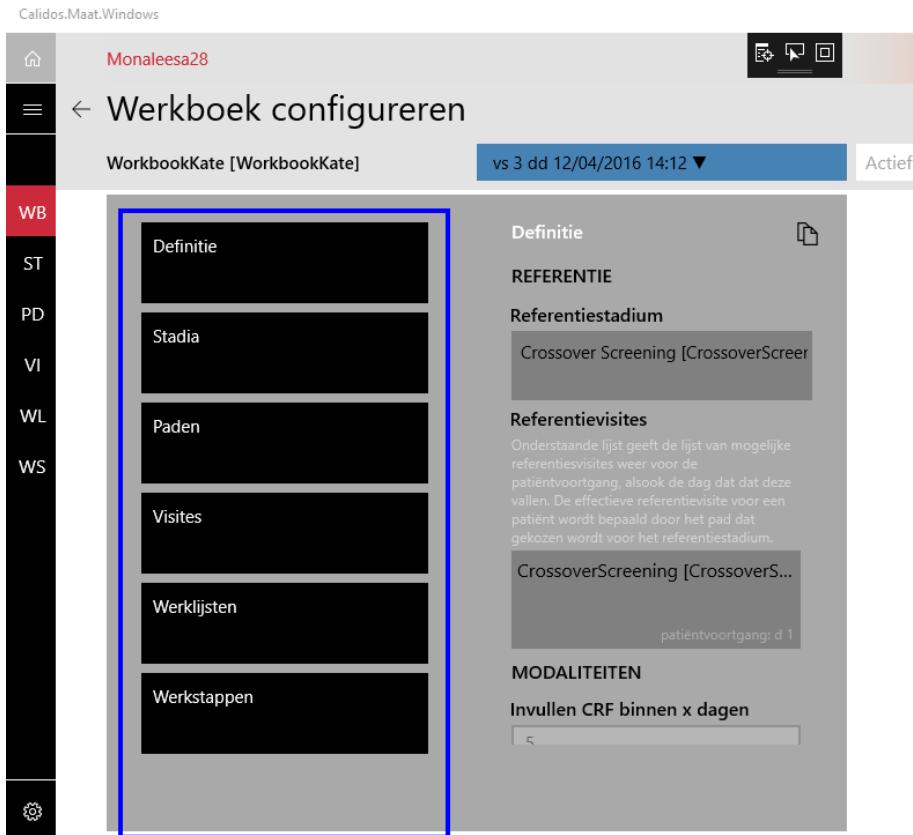
## UseOptionsListViewControl

This function is designed to test if clicking one of the options on the left side of the screen results in the correct list appearing to the right of it. The instance is the number of the option in the view (first, second, third,...)

```
public void UseOptionsListViewControl(int _instance, Action _action)
{
    XamlControl _optionsListView = UIMapConfig.UICalidosMaatWindowsWindow.UIHub.UIHubSection.UIOptionListView;
    XamlControl _selectedOptionsListView = UIMapConfig.UICalidosMaatWindowsWindow.UIHub.UIHubSection.UISelectedOptionListView;
    XamlControl _selectedOptionsTitleText = UIMapConfig.UICalidosMaatWindowsWindow.UIHub.UIHubSection.UISelectedOptionTitleText;
    XamlControl _control = GetChildFromList(_optionsListView, _instance);

    UseConfigListViewControl(_control, _action);
    if (_selectedOptionsListView.Top < 0 || GetChildFromList(_control, 0).Name != _selectedOptionsTitleText.Name) Assert.Fail("Clicking
```

With options, we mean the following:



#### ChooseWorkbookConfigWorkVersion

A function designed to choose the latest work version to test on, because otherwise some options will be disabled. Note that this is a hardcoded function, and if something should change to the version-selection button, the function will not work anymore. The reason this function is hardcoded is that we cannot access the version-options with the coded UI test builder.

```
public void ChooseWorkbookConfigWorkVersion()
{
    UseControl(UIMapWorkbookConfig.UICalidosMaatWindowsWindow.UISelectVersionButton, Action.Click);
    Playback.Wait(DelayMilliseconds);
    Keyboard.SendKeys("{Tab}");
    Playback.Wait(DelayMilliseconds);
    Keyboard.SendKeys("{Up}");
    Playback.Wait(DelayMilliseconds);
    Keyboard.SendKeys("{Enter}");
}
```

### UseConfigListViewController

This function navigates to one of the config-pages. You just have to give the control and the execution-method to it, the function does the rest.

```
public void UseConfigListViewController(XamlControl _control, Action _action)
{
    if (_action == Action.Tab)
    {
        _control.WaitForControlExist();
        TabToControl(_control);
    }
    if (_action == Action.Arrows)
    {
        _control.WaitForControlExist();
        UITestControl _parent = _control.GetParent();
        UseControl(_parent.GetChildren()[0] as XamlControl, Action.Click);
        GoToControl(_control);
    }
    else if (_action == Action.Click)
    {
        _control.WaitForControlExist();
        Mouse.Click(GetMiddleOfControl(_control));
    }
    else Assert.Fail(_action.ToString() + " is not possible on this control!");
}
```

## Automation of Coded UI Tests and its results

### Automation: Adding Coded UI Tests to the build street

After looking into a lot of options the main conclusion here is that, for now, adding the coded ui tests to the build street is not possible. The main reasons for this are:

- TFS 2013 cannot run a virtual windows 10 environment to deploy the application to
- TFS 2013 only supports windows 8 environments and lower to use with test agents

While TFS is not upgraded to a newer version, the testing automation will only go as far as to manually start the testrun, then leaving the computer to AFK-test for that time. In the end the test run will produce a TRX file that contains all details concerning failed, passed, aborted or not executed tests.

However, should there be an upgrade to the TFS 2015 or cloud version in the near future, this link might provide a good start to setting up the automation: [Specify the steps of your build on VSTS & TFS](#)

### Result Management Tools: Visualizing results

Coded UI Tests can help out with a lot of normally manual testing on your application. It's an easy - and timesaving - way of discovering problems, bugs or other unwanted things within your app. But while automated Coded UI Tests do all that work for you, you still don't have a full view of the status of testing the many facets your application should be tested upon. Therefore, we designed the Result Management Tools. These handy tools bring all the result data together in one HTML file, so to give the developer/tester a better view of the testing progression.

There are 2 Tools (or 3 if you count the HTML file too) that make all this possible, in combination with 3 XML files. A detailed explanation can be found in the next paragraphs.

#### Tool: TRX 2 XML Parser

As easy as it says. This tool will provide a fast way to reduce the redundant information the TRX file contains, and put all useful information into the newly generated XML file. As said in previous paragraphs or chapters, the TRX file contains test run data. This is a lot of data - believe it or not - but not all of it is useful for the developer/tester.

#### RegisterTest

The first thing you have to know about the TRX file is that, for this tool to work, we need to add another thing to every coded ui test that has been created. A reference as to what it is actually testing. To indicate this we designed the RegisterTest method.

The RegisterTest method - defined in BaseTestClass - **needs to be called in every test method as the first line** of that test method. This is done **so to automatically check if a test is completed**, failed or not run at all. The method will write more details into the file so to know exactly where the tests are coming from (page/control and paradigm). To specify the page/control and paradigm we use GUID's (Global Universal Identifiers).

## Running tests

A test run can be started in the CMD for developers from visual studio, by using the .dll of the Calidos.Maat.CodedUITests project. This can be done when in the bin/debug (or in this case DevAzure) folder and calling following line:

"MSTest /testcontainer: Calidos.Maat.CodedUITests.dll"

- MSTest The program that will be used to start the testing
- testcontainer The file that is used to search for test methods that can be run

> **Note:** To change the folder to which the command prompt window opens by default, on windows, choose **Start**, point to **Microsoft Visual Studio 2015**, point to **Visual Studio Tools**, right-click **Developer Command Prompt**, and then choose **Properties**. In the **Developer Command Prompt Properties** dialog box, you can change the path to the default folder in the **Start in** box.

Another option to start the test run is to just open your project in Visual Studio, select **Test**, then choose **Run**, and eventually click **All Tests**.

When tests are done running the program publishes a logfile in the bin/DevAzure folder under the new folder named "TestResults". More information about the TRX file and where it's deployed, or where you could deploy it to can be found in following link: [How to: Deploy Files for Tests](#)

## Running the TRX 2 XML Tool

To run the TRX 2 XML application, it was necessary at first to add command line arguments under the properties tab. This will reference the target and input file paths. This will be automated in the near future.

"Command line arguments" during testing at this time:

-  
TestFilePath="C:\Dev\CTO\Src\Dev\Jasper.0\Client\Calidos.Maat\Calidos.Maat.CodedUITests\bin\DevAzure\TestResults\Jasper\_VIRAPTORUS 2016-04-19 12\_37\_25.trx" -Output="C:\Temp\"

When using the application it will recover the file (in this test it will be from the gray colored file path). It will execute all code and parse it into an XML file and put it in the designated folder (in this test it will be in the red file path)

## Result XML

When we have the complete TRX file we can parse it with our TRX 2 XML tool that will handle the parsing process from the TRX to an XML file - known as the Result XML - with the use of a streamwriter and the System.Xml.Linq class.

The tool will search for the UnitTestResult tag within the TRX and pick up the necessary data to write into the XML. In this case, "outcome" and "testName" Then searches for the DebugTrace tag to find the data inserted by the RegisterTest method. Every value that contains the "MaatTest]" indicator will be obtained and also added to the XML file. The Result XML will have following syntax:

```
<Test TestId="Test Name" CategoryId ="Category GUID" ObjectId="Object GUID" ResultLabel="Outcome">
```

- TestId
  - The test name (test method name) that was given in the code of the test
  - CategoryId
  - The ID of the category, written as a GUID
  - ObjectId
  - The ID of the object, written as a GUID
- ResultLabel
  - Contains one of 4 different possible statuses:
    - Passed      The test passed
    - Failed      The test failed
    - Aborted      The test was being executed but got manually aborted
    - Not Executed The test has not been executed at all

#### Tool: TestResultParser

The TestResultParser, in short, accepts 3 XML input files and creates an HTML file as its output. The first XML file we already discussed. In this section we will discuss the other 2 XML files and the end product, the HTML file, as well as how this tool actually works.

#### HTML file: Result Table

To understand what kind of input we need or use, we have to start at our end product. The HTML file will give a visual representation of the progress considering test completion of your application. This is done through a biaxial table or in other words a matrix. In our coded ui testing there is an axis that represents paradigms and one that represents controls/screens. A cell in the matrix is the junction of each paradigm and control/screen, or in better words a cell represents a test case. A test case will always show its percentage passed or in some cases the letter that represents if it is To Do (T), Not To Do (N) or Unknown (-). Each cell also contains one of 5 different states of testing:

- 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

As for cells always showing their total percentage passed. This means they also contain the percentage passed of their children. Even tho sometimes it occurs that a parent cell also is a test case itself. In this case, The cell shows an addition of both the test case's own percentage passed and the percentage passed of its children

> **Note:** When fixing test cases, it's good practice to always start fixing the lowest descendant of the parent that indicates a 'Failed' status. This to make sure the parent is, or is not, the problem of the 'Failed' status.

### Definition XML

The Definition XML is used to manually define the axes of the table. Remember when we talked about GUIDs? Well here's where we define them along with an appropriate name for each paradigm and screen/control. The syntax has a CategoryDefs and ObjectDefs tag that contain their Objects/Categories that are defined as follows:

```
<Category id="Category GUID" name="Paradigm" info="Description" level="">  
<Object id="Object GUID" name="Screen / Control" info="Description" level="">  
    • CategoryId / ObjectId  
    ○ The ID, written as a GUID  
    • Name  
    ○ Paradigm or screen/control name  
    ○ Require to be unique  
    • Level  
    ○ The level object will be calculated and therefore doesn't need to be defined  
      because it will be overwritten in any case.
```

### Target XML

The Target XML manually defines a target for each test case. The syntax is as follows:

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

- TargetName
  - Descriptive name of the category and object (in readable text)
  - This, to help understand what is exactly there (GUID only is hard to read)
- CategoryId
  - The ID of the category, written as a GUID
- ObjectId
  - The ID of the object, written as a GUID
- TargetLabel
  - Contains one of 4 different possible statuses:
    - To Do This test case needs 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
    - Done This test case is manually acknowledged to be 100% completed

### XSD Files

XML Schema Definition in full, once again a name that says it all. Every XML has an XSD behind them, mainly for one simple reason. We use Xsd2Code++ to map all defined objects in XML to objects in Visual Studio.

## Running the TestResultParser tool

The tool executes a number of methods that result into generating HTML for the HTML file. Let's have a short look at them

First we create our full matrix in memory using three methods:

- ProcessCategories
- ProcessObjects
- CreateCells

### ProcessCategories

- Creates a list of all categories for the vertical axis
- Calculates the level of each category
- Uses the Definition XML to obtain the right properties

### ProcessObjects

- Creates a list of all objects for the horizontal axis
- Calculates the level of each object
  - Uses the Definition XML to obtain the right properties

### CreateCells

- Creates all the cells (a total of: Horizontal Axis \* Vertical Axis)
- Maps the parents and children correctly so to create totals

When the empty matrix is generated it gets time to populate it. This is where our Target and Result XML files come in.

### ProcessTargetData

- Maps Target XML data to the correct cells using the GUIDs and the TargetLabels

### ProcessResultData

- Maps Result XML data to the correct cells using the GUIDs and the ResultLabels

When all the virtual generation of the table is completed we can start writing the HTML code for it. The HTML is built in following order:

- ProduceHtml
  - CreateHtmlHead Create metadata
  - CreateHtmlBody Create Table, Legend, Failed list
    - CreateTestResultDiv Create Table wrapper
      - CreateTestResultTableHead Create Table Head (Objects/Screens)
      - CreateTestResultTableBody Create Table Body (Cells / Categories)
    - CreateLegendDiv Create Legend in wrapper
    - CreateFailedDiv Create Failed Test List in wrapper

# Log

## Coded UI Testing problems

### Navigation looping (Testing click, enter and space in 1 method)

- Failed:
  - Looping over an array of actions in the base-class methods for navigation, because testInitialize is not repeatedly executed.
  - Test methods within test methods
  - Executing the method 3 times and executing testInitialize after every time, all inside one test method

### Content - Read

- When the found patient list is too long, the test fails after 4 minutes even though all search results are correct.
- ClinicSearch

### Functionality

- Cannot (yet) access elements of a flyout button
- Partpicker (date/time), can access the yes/no button with "Ctrl-i" but cannot access the other controls in the popup-window.

### General random fails

- Beveiligd geheugen:

Calidos.Maat.CodedUITests.Screens.Config.WorkbookConfigPage.WorkbookConfigPageContent.WorkbookConfigReadMenubarWorkbookConfigToggleButton threw exception:

System.AccessViolationException: Poging tot het lezen of schrijven van beveiligd geheugen. Dit duidt er vaak op dat ander geheugen is beschadigd.

- NullReferenceException:

Calidos.Maat.CodedUITests.Screens.Config.WorkbookConfigPage.WorkbookConfigPageNavigations.WorkbookConfigNavigateContactConfigListItemClick threw exception:

System.NullReferenceException: De objectverwijzing is niet op een exemplaar van een object ingesteld.

- ElementNotFoundException:

Calidos.Maat.CodedUITests.Screens.Config.WorkbookConfigPage.WorkbookConfigPageNavigations.WorkbookConfigNavigateTrialHyperlinkEnter threw exception:

System.Windows.Automation.ElementNotFoundException: Element not available --->

System.Runtime.InteropServices.COMException: Een gebeurtenis kan geen van de abonnees aanroepen  
(Uitzondering van HRESULT: 0x80040201)

*This exception is thrown when the parent-control of the control you want to use has disappeared. So one possible explanation for this problem could be that the test is trying to find a control before the page has been refreshed (due to appStartUp), and has already found the parent-control but then, the page refreshes and the originally found parent control has therefore disappeared.*

*After testing, we came to the conclusion that putting in a delay before searching the first control may have some positive effects on the testresults (less fails) but does not fix the problem 100%.*

- System.Runtime.InteropServices.COMException:

Calidos.Maat.CodedUITests.Screens.Config.WorkbookConfigPage.WorkbookConfigPageNavigations.WorkbookConfigNavigateTrialHyperlinkSpace threw exception:

Microsoft.VisualStudio.TestTools.UITest.Extension.UITestControlNotFoundException: The playback failed to find the control with the given search properties. Additional Details:

TechnologyName: 'UIA'

FrameworkId: 'XAML'

ControlType: 'Hyperlink'

---> System.Runtime.InteropServices.COMException: Fout HRESULT E\_FAIL is gereturneerd voor een aanroep van een COM-onderdeel.

- System.ComponentModel.Win32Exception:

Calidos.Maat.CodedUITests.Screens.Config.WorkbookConfigPage.WorkbookConfigPageNavigations.Initialize threw exception. System.ComponentModel.Win32Exception:

System.ComponentModel.Win32Exception: Kan RPC niet uitvoeren.

## Problems / Functionality issues

### General

- Login window pop-up window interrupts test startup, causing them to fail even when there is no problem with the UI or test itself
- Pop-up error window interrupts tests, causing them to fail even when there is no problem with the UI or test itself
- Settings Button on Menubar does not have navigation (yet)
- When "Mäat" is not yet active and navigating directly towards a pagetoken (with extra parameters) for Testing purposes for example, the page sometimes gives an error and/or is not fully loaded (Ex. PatientScriptPageStates)

## Page-specific

- Some navigation controls cannot be activated using 'Enter' or 'Space'
  - ClinicHub
    - VisitesEnFollowUpsHub: all tiles
  - TrialHub
    - StudieInfoHub: all tiles
    - StudieVerloopHub: all tiles
    - PatiëntPraxisHub: all tiles
    - VisitesEnFollowUpsHub: all tiles
    - NotasHub: tile
- SemanticZoom tiles spaced wrong / not scrollable when in zoomed out mode
  - ClinicHub
- Clicking a trial and then clicking on the BackButton sometimes makes a hyperlink appear on top of the page of the trial you just navigated to, but sometimes it doesn't
  - ClinicTrials
- PatientScript-worklist:
  - worksteps disappear after a while and are replaced by: "Geen werkstappen in deze werklijst"
  - SemanticZoom:
    - work different on some pages, instead of being made of Hubs and HubSections, they are sometimes groups. Because of this, we cannot always use our own BaseClassCodedUI methods
    - Because the list inside the SemanticZoom doesn't have a UID, we have to give the searchproperties an instance (instance 2) to find the correct list, otherwise the list found in the testmethod will be the tab-list on top of the page, which does contain a UID
- WorkbookConfig
  - Definitie:
    - textfield below "Modaliteiten" is only partially displayed
- PathConfig
  - When navigating to PathConfig starting from WorkbookConfig, the ProgressBar keeps loading and no control is accessible, the full page is frozen

# Possible visual or interfering issues

## General

- Home button toggle isn't working as intended
  - Still toggles if user is already at homepage
- Buttons that are actual XAML buttons (black) have bad contrast when highlighted
  - Text turns black on black background when mouse hovers the element
- ClinicHub Hubs that contain lists get a different interaction animation with their respective hub titles
  - Probably because they make the width of the button bigger and hereby cause the different interaction
- Visites & Follow-ups is referenced differently on different pages
  - Sometimes 'Visites & Follow-ups' (ex. in ClinicContacts as title of page)
  - Sometimes 'Visites en follow-ups' (ex. in TrailHub as title of HubSection)
  - Uniform naming could improve the testing possibilities
- When selecting a hubsection title with tab and activating it using the enter or space key, the corresponding tile with that title is not highlighted.
  - By default the program highlights the first tile, is this logical?

## TrialHub

- Press any ListItem (preferably more at the back for demonstration purposes)
- Use the back button
  - We end up at TrialHub, but at the start
  - Not on the respective hub that was clicked

## ClinicContacts

- "Visites" Combobox items sometimes contain small logos. These heighten the combobox when selected.
  - Not sure if this is intended
- "Visites" Combobox items contains "Alle visites" twice

## ClinicContactsPage

- Use combobox to order chronologically
- Click any ListItem
- Click 'back' button
  - We end up at ClinicContactsPage, but ordered by task
  - Not ordered chronologically

## PatientHub

- When clicking “Casus info”
  - “Administratief” and “Status” tabs overlap (text)

## ClinicTrials.NewTrial

- Typing wrong data makes a sentence appear “Gelieve een geldig studienummer in te geven”, however, the sentence is not fully displayed

## **Fixes**

### Adding controls to the Ulmap

- When a title is variable (Ex. title of a trial), we have to find a way to always be able to search the title, based on some other properties. Because the title doesn't have an automationID, we search for the title based on an hard coded instance, but every time something is added in the titlebar, we have to change the instance.

### Adding popup menu's to the UI Map

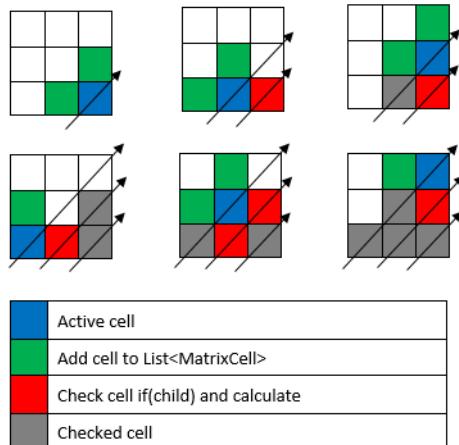
- Use “Ctrl-i” while hovering over the popup menu to directly select it with the coded UI test builder, without having to drag the crosshair onto the control (in that case, it disappears)

## **Result Management Tools**

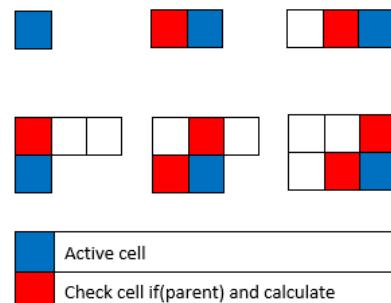
### Calculating totals algorithm

- Algorithm 1
  - The first algorithm was too slow, probably because of LINQ queries
  - Inheritance coupling based on level of each cell left and top of current cell
  - Totals calculated after matrix was already in memory
  - Not efficient on small or large scale data processing
- Algorithm 2 (Revised Algorithm 1)
  - LINQ has been removed from the equation
  - Inheritance coupling based on level
    - Hold a list of children and a parent object in the created cell
    - Totals are calculated ‘on the fly’
  - Performance rate increased by  $\frac{1}{3}$ 
    - Program run time now lasted 10s instead of 30s
  - Very efficient on large scale data processing

## Algorithm 1



## Algorithm 2



## Generating HTML

- Before
  - After swapping the test data to 'real' data, the HTML generation was rather slow
    - Program run time back to 30s
  - Iterated over both VerticalAxis and HorizontalAxis
  - Used a LINQ query along with a list that was passed to AssignClassBasedOnLabel() method
  
- Revised methods
  - Removed LINQ query and List
  - Changed AssignClassBasedOnLabel() method to accept only 1 cell at a time
  - Iterates over every cell in matrix.Cells List
  - Counters → to keep up with which cell's data is being altered

# Rapport Calidos

## UID's

To be able to use a control in a test, the control has to be added to a UIMap. This is a document which contains a hierarchy of all controls of a page. When a control is added to this map, you can generate it in code by writing the hierarchical path from the top parent down to the control you need.

Example:

```
XamlButton ValidateOverlayButton =  
UIMapWorkbookConfig.UICalidosMaaatWindowsWindow.UICommandBarMainActionCustom.UIValiderenButton;
```

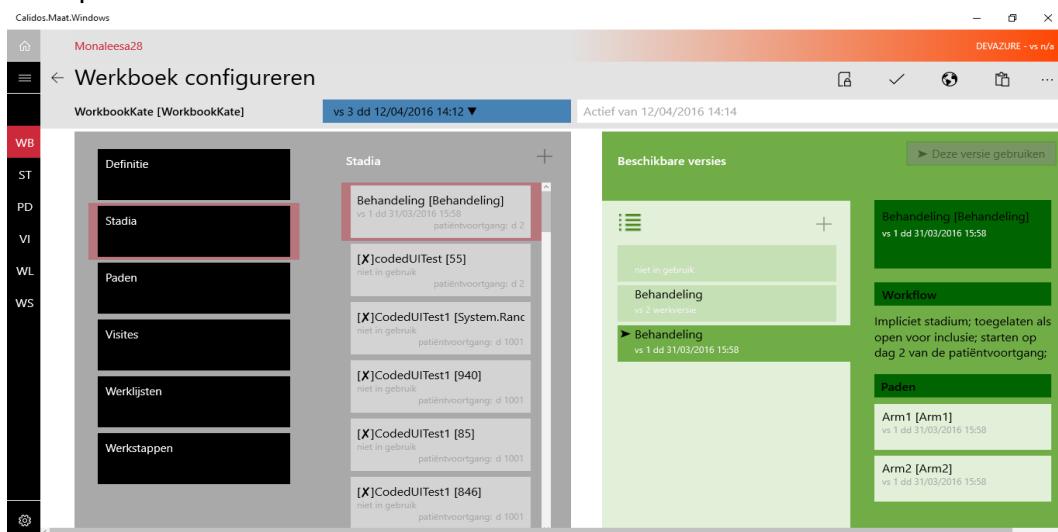
Because some controls do not contain a UID (unique identifier) to search on, the UIMapping sometimes goes wrong. For example, when there are several identical controls displayed, all on the same hierarchical level, but they do not contain a UID, they are mapped by instance. The elements inside these controls will all be mapped to the first control, even if they exist inside the second control. There are other possible scenarios in which the UIMapping goes wrong, but this one is the most common. And all scenarios have something to do with UID's.

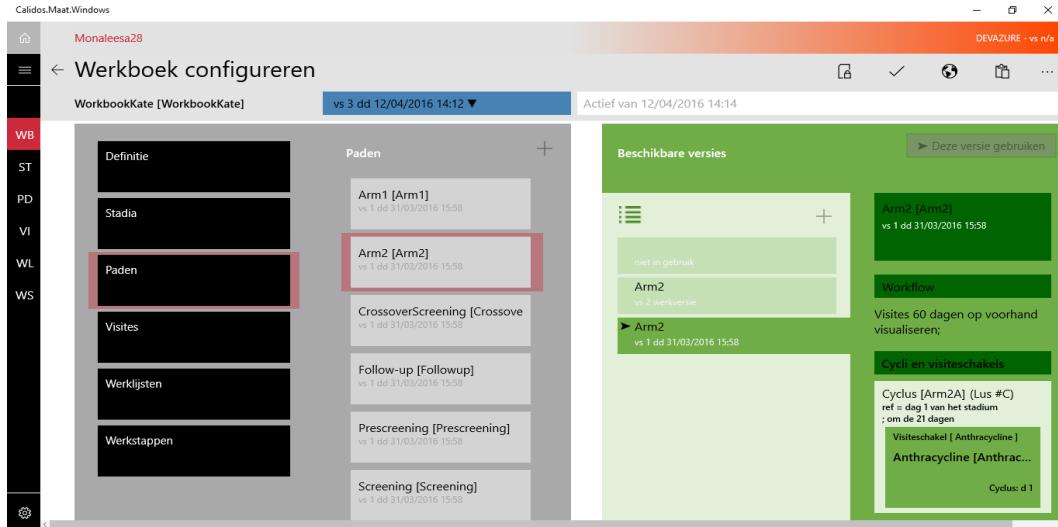
Because of this bad UIMapping, the tests often require impractical workarounds to test some controls. A solution would be to make sure every single element is given a unique UID. When this is correctly done, the hierarchical structure should be correctly mapped.

## Xaml paradigms

We found that sometimes, elements that look the same and act the same visually, still have a different hierarchical structure or different algorithms. To make the application easier to test, it would be useful to create some sort of guideline for every paradigm, explaining a set of coding rules to which the code should conform for that paradigm.

example:





Look at the controls on the far right, starting from the first control containing the main info of the version, with below it the workflow and below it a list of some other elements. Although this set of controls look the same on 2 different tabs, and have the same functionality, they are differently mapped. On the top screen (stages), every control I just described (interpreting the list on the bottom as one control) exists separately as a child-control of the complete hubsection. However on the bottom screen (paths), the complete set of controls just described, exists inside a list, which is a child of the hubsection.

Another good example is the semantic-zoom functionality. On some pages, the hiérarchical structure of the elements inside the semanticzoom is different from other screens, even though the paradigm stays the same.

## Random failing tests

We noticed that even when tests are written completely correct, there are still some tests failing when executing them all together (in a sequence). One possible explanation for this problem could be that the test is trying to find a control before the page has been refreshed (due to appStartUp), and has already found the parent-control but then, the page refreshes and the originally found parent control has therefore disappeared.

After testing, we came to the conclusion that putting in a delay before searching the first control has a positive effect on the amount of tests that fail. However we cannot guarantee a 100% that all tests will pass.

Note that this delay-time is defined in our BasClassCodedUI, so when you want to lengthen or shorten the delay-time, you can do this by changing it only once.

# Future work (What was planned)

TODO:

## TestResultParser

- Revise 'Failed Tests' div in HTML file
  - Only shows tests that have ran and got an outcome but have no cell to map their data towards
- 'Done' labels don't add up as they should
  - Needs a fix at MatrixCell's CalculatedValue
- Adding floating headers to HTML file
- 'info' tag in CategoryDefs and ObjectDefs is redundant
  - Check out ProcessCategories and/or ProcessObjects method(s))
  - 'Info' tag is used, but the defined value is overwritten/ignored
- 'level' tag in Definition XML is redundant
  - Gets automatically calculated by the system and should not be defined empty

## Testing Maät

- Workbookconfig
  - +-40% tested
  - Up to date according to latest rules (baseclass per page etc.)
- All other tests:
  - Outdated according to latest rules
  - Outdated tests
  - → Re-analyse and adapt where necessary
  - Analyse all other pages & start writing tests
- Manual checklist:
  - WorkbookConfigPage-checklist = up to date
  - Paradigm checklist = up to date
  - All other checklists: Re-analyse & adapt where necessary

# Time estimation

To estimate the amount of time it would take to test the full application, we estimated the time it would take to test each page, based on our experience with the already tested part of the application. We then added all these numbers together and divided them by 2, assuming the test process would speed up or be used iterative after a while. Here are the calculations:

## Analysation of all pages

Amount of pages	Type	Amount of days for 1 page		Total days needed
6	Config page	14	$6 \times 14$	84
12	Complex page	6	$12 \times 6$	72
10	Normal page	3	$10 \times 3$	30
3	Simple page	1	$3 \times 1$	3
5	Easy page	0.5	$5 \times 0.5$	2.5
7	Unknown	4	$7 \times 4$	28
	Buffer Time	2	$1 \times 2$	2
			Total	221.5

Be aware that this is only an estimation, we cannot possibly consider every possible parameter.

## Result

221.5 days for one man to complete the full application on his own.

It would result in around 110 days for 2 people to completely test the full application

## Conclusion

It's a lot of work to completely test the application, but that's no reason to not put time and effort into it. Much easy-to-miss bugs can be detected this way. Also keep in mind that this application is rather large in size not only data-wise. A lot of controls as well as a large amount of data means a lot of time to be invested into the page.

# Windows Store for Business (SFB)

- A. Microsoft Store
- B. Store for Business
- C. ~~Link PC met Store for Business~~
- D. ~~Meerdere Stores van verschillende bedrijven op 1 PC~~
- E. Version upgrading
- F. ~~Power Shell??~~
- G. Windows Intune

- Probleem: Heb geen Development Account om in het Windows Dev Center in te loggen en aanpassingen te maken
- Cruciaal om sommige dingen te bevestigen dat deze effectief werken
  - o App in Windows Store zetten
    - Package van app maken en door tests laten gaan is al gelukt, enkel deploy vereist een dev account
  - o LOB publisher zijn voor Store for Business
    - Om te zien hoe dit verhaal het probleem zou kunnen oplossen

## A. Microsoft Store

- App creëren
  - o Noodzakelijk om images (assets) aan te passen zodat deze niet 'default' zijn
    - Belangrijk om door tests te komen (i.v.m. branding,...)
  - o Laten runnen in 'release' op Local Machine (deploy locally)
  - o Pakketten creëren
    - Rechterknop op project file in solution explorer > Store > Create store packages
    - Dialoog pop-up
      - Selecteer 'Yes' bij eerste scherm
      - Vul je developer account gegevens in
      - Eventueel velden aanpassen waar nodig > Create packages
    - Nieuwe pop-up
      - Eventueel velden aanpassen waar nodig > Launch Windows App Verification Kit
  - o Verificatie van App
    - Duurt even → gewoon pc laten doen
    - Pakketten zijn gegenereerd
      - Bevat een 'upload' bestand om naar de Windows Store te deployen

## B. Store for Business

- Store for Business opzetten:
  - o Azure Portal
  - o Azure AD aanmaken
  - o Users toevoegen aan domein
    - Jezelf Global Administrator maken → Beheer / opzetten van Store for Business
    - Andere users aanmaken
  - o Surf naar <https://www.microsoft.com/nl-be/business-store> en klik “Aanmelden met een organisatieaccount”
    - Global Administrator adres gebruiken om aan te melden en “Store for Business” op te zetten.
  - o Beheer kan makkelijk
    - Apps kunnen rechtstreeks ge’assigned’ worden aan personen die in de AD een user zijn.
  - o Na ongeveer 12 uur is de app beschikbaar voor de gebruikers die in de AD opgezet zijn, en hierdoor door de Global Administrator de rechten hebben gekregen deze te zien/gebruiken
    - Inloggen in Windows Store onder de gebruiker en een extra tab wordt zichtbaar
      - Bij mijn tests → Naam van tab = Directory-naam op Azure portal
      - o In mijn geval: “JasperVanGestelHotmailCom”
- Line-of-Business (LOB):
  - o Specifieke publisher in staat stellen zijn apps vrij te geven voor bepaalde bedrijven
  - o Via Store for Business en Windows Dev Center
    - Store for Business
      - Uitnodiging sturen naar publisher om als LOB publisher voor jou bedrijf toe te treden

## C. Link PC met Store for Business

/

## D. Meerdere Stores van verschillende bedrijven op 1 PC

/

## E. Version upgrading

- **Up-to-date apps** - The Store for Business manages the update process for apps with online licenses. Apps are automatically updated so you are always current with the most recent software updates and product features. Store for Business apps also uninstall cleanly, without leaving behind extra files, for times when you need to switch apps for specific employees.

## F. Power Shell??

/

## G. Microsoft Intune

Belangrijkste te onthouden uit Intune features en mogelijkheden:

- Met Intune beperkt u de complexiteit om de bedrijfsgegevens te beveiligen tot een minimum, door het mobiele apparaatbeheer via de cloud met geïntegreerde beveiligings- en nalevingsfuncties.
- Een selfservice bedrijfsportal maken waar gebruikers hun eigen apparaten kunnen inschrijven en bedrijfstoepassingen op de populairste mobiele platforms kunnen installeren.
- Wanneer een apparaat is ingeschreven, kunt u automatisch certificaten en WiFi-, VPN- en e-mailprofielen instellen, zodat gebruikers met de juiste beveiligingsconfiguraties toegang kunnen krijgen tot bedrijfsbronnen
- Geef uw medewerkers de mogelijkheid veilig toegang te krijgen tot bedrijfsgegevens op uw bestaande bedrijfstoepassingen met de Intune app wrapper, zonder dat u code hoeft te wijzigen in deze toepassingen
- Via het selfservice bedrijfsportal beheerders in staat stellen vereiste apps tijdens de inschrijving te pushen en gebruikers toestaan bedrijfsapps te installeren
- While software deployment has always been a challenge across larger organizations, the process of deploying software across the Internet is significantly more complex. The Windows Intune **Managed Software** and **Updates** workspaces simplify how your business can deploy and manage software and 3rd Party updates across your managed computers, wherever they are.
- Voor de volle informatie, zie URL's bij "sources"
- Volgens de site ligt de kostprijs van "Intune" rond de 5 euro per maand per gebruiker

## H. Conclusies

- Store for Business is in zeker zin een feature van de Windows Store
  - o Implementeert dezelfde werkwijze als de Windows Store, met enkele voordelen specifiek voor een enkel bedrijf.
  - o 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'
  - o 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
  - o 'instellingen' typen in Windows search > Bijwerken & beveiligen > voor ontwikkelaars
  - o Side-loading kan ook via deze weg ingesteld worden, aangezien er een 'sideload modus' beschikbaar is onder deze tab
- **Line-of-Business** lijkt een mogelijke oplossing voor het gestelde probleem
  - o Via Mobile Device Management (**MDM**) zoals Windows Intune lijkt het mogelijk om updates gesynced te houden met alle devices
  - o LOB kan via de Windows SFB
  - o **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.
        - o Enkel de ingeschreven devices zijn in staat deze te installeren
        - o 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.)

# I. Sources

Publishing to Store

<https://msdn.microsoft.com/en-us/library/windows/apps/hh454036.aspx>

Info Store for Business

<https://technet.microsoft.com/en-us/windows/store-for-business.aspx?f=255&MSPPError=-2147217396>

Info LOB → publishing apps as in house or third party developer directly to the SFB

<https://blogs.windows.com/buildingapps/2015/11/16/increase-your-apps-reach-with-windows-store-for-business/>

More info on LOB and SFB

<http://www.comparex-group.com/web/com/about/press/2016/setting-up-a-store-with-microsoft-store-for-business.htm>

Why LOB is VERY interesting → considering version updates (via MDM's)

<https://technet.microsoft.com/nl-nl/library/mt592935%28v=vs.85%29.aspx>

SFB Video (Video)

<https://channel9.msdn.com/Events/Windows/Developers-Guide-to-Windows-10-Version-1511/Windows-10-for-Business-Publishing-apps-to-the-Business-Store>

Windows Intune – features of MDM

<https://www.microsoft.com/nl-be/server-cloud/products/microsoft-intune/features.aspx>

Windows Intune 3d Party Software (Video) (Starting around 10:00 min)

<https://www.youtube.com/watch?v=Hk8OmXWbG30>

Deploying Software and Third Party Updates with Windows Intune

<https://technet.microsoft.com/en-us/library/hh441740.aspx>