Voortgangsverslag 1 - *Jasper Van Gestel*

Coded UI Testing:

Promotors

- **Stagebegeleider:**
  - Naam: Patrick Van Houtven
  - E-mail: patrick.vanhoutven@ap.be
  - Gsm: 0474 66 57 15
- **Stagementor:**
  - Naam: Mark Devos
  - E-mail: Mark.devos@calidos.be
  - Gsm: 0475 69 13 42
- **Stagementor:**
  - 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*.

Aanvankelijk is het de bedoeling om te leren werken met de *'Coded UI Test Builder'* van *'Microsoft Visual Studio'*. Dit aan de hand van kleine projecten die zelf worden opgezet of al reeds bestaan. Verder is het de bedoeling om het gehele proces zo automatisch mogelijk te maken.

Er wordt verwacht dat er een *'Guideline voor de tester'* zal zijn 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.

Technische omschrijving

Guidelines

Outlines

Windows 10 Metro Applications

To get a better view and idea on what we are doing with our project, we will take a closer look into the platform that we are using. The Windows 10 Metro App make it easy to build a single project that runs on all supported platforms, which will include Windows 10 PCs, tablets, phones, Xbox One and Windows 10 IoT (Internet of Things).

Previously, a Universal App included multiple projects, one for shared code, and another for each supported platform, and you would build separate apps per platform. In order to achieve this level of compatibility, Microsoft has factored out a “Windows Core” which is common to all platforms.

UWP

Designing an app that looks good on such a wide variety of devices can be a big challenge. So how do you go about designing an app that provides a great UX on devices with dramatically different screen sizes and input methods? Fortunately, the Universal Windows Platform (UWP) provides a set of built-in features and universal building blocks that help you do just that.

UWP app features

- Effective pixels and scaling
    - UWP apps automatically adjust the size of controls, fonts, and other UI elements so that they are legible on all devices.
- Universal input and smart interactions
- Universal controls
    - The UWP provides a set of universal controls that are guaranteed to work well on all Windows-powered devices. This set of universal controls includes everything from common form controls like radio button and text box to sophisticated controls like grid view and list view that can generate lists of items from a stream of data and a template. These controls are input-aware and deploy with the proper set of input affordances, event states, and overall functionality for each device family.
- Universal styles
    - Your UWP app automatically gets a default set of styles that gives you a bunch of different features

Responsive design techniques

- Reposition

    - You can alter the location and position of app UI elements to get the most out of each device. In this example, the portrait view on phone or phablet necessitates a scrolling UI because only one full frame is visible at a time. When the app translates to a device that allows two full on-screen frames, whether in portrait or landscape orientation, frame B can occupy a dedicated space. If you're using a grid for positioning, you can stick to the same grid when UI elements are repositioned.

- Resize
    - You can optimize the frame size by adjusting the margins and size of UI elements. This could allow you to augment the reading experience on a larger screen by simply growing the content frame.

- Reflow
    - By changing the flow of UI elements based on device and orientation, your app can offer an optimal display of content. For instance, when going to a larger screen, it might make sense to switch larger containers, add columns, and generate list items in a different way.

- Reveal
    - You can reveal UI based on screen real estate, or when the device supports additional functionality, specific situations, or preferred screen orientations.

- Replace
    - This technique lets you switch the user interface for a specific device size-class or orientation. In this example, the nav pane and its compact, transient UI works well for a smaller device, but on a larger device tabs might be a better choice.

- Re-architect
    - You can collapse or fork the architecture of your app to better target specific devices. In this example, going from the left device to the right device demonstrates the joining of pages.

Testing: Basics

Hierarchical navigation design

In this section, you can see a hierarchical design of all possible navigations inside the Clinical Trials application, ordered by page.

ClinicHub

PatientHub

TrialHub

WorkBookConfig

Method Naming

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

- 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 tot 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

The Clinical Trials application follows a specific map structure for all it’s elements. To keep the file structure readable, we follow the same structure inside our testproject.

- Calidos.Maat.CodedUITests
  - Screens
  - General UI Map
  - Partial parent .cs files
  - '.csv' global scenarios
  - Category (clinic,...)
    - '.csv' categorical scenarios
    - Page (contactspage,....)
    - UI Map

        - '.csv' local scenarios
        - '.cs' files per category
            - States
            - All tests regarding the different states of a page
            - Navigation
            - All tests regarding forward navigations from a page and backwards navigation to the page
            - Functionality
            - All tests regarding functionality/interfunctionality of controls
            - Content
            - All tests regarding data in the database

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.

Content

Create

- 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.

Read

- Check if the data within the control is correct

Update

- Check if changing data inside the control makes the data change where it should change elsewhere

Delete

- Check if deleting specific data triggers the dissapearance of the data elsewhere in the application

Custom

- Page specific functionality regarding data in the database

Navigations

- Check if we navigated to the right page when control is activated
- Check if navigation triggered the correct page-state

States

Semantic zoom

Zoom out

Ctrl-

- 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.

Ctrl & mousescroll

- 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.

Tab/arrow select

- Check if using the tab key (or arrow keys) selects the hubsection titles in the semantic zoom and using a specific key makes the semantic zoom zoom out.:
  - Space key
  - Enter key

PgUp/PgDn select

- Check if using the PgUp / PgDn keys selects the hubsection titles in the semantic zoom and using a specific key makes the semantic zoom zoom out.:
  - Space key
  - Enter key

Mouseclick

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

Zoom in

Ctrl+

- 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).

Ctrl and mousescroll

- 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).

Tab/arrow select

- See 'Zoom out'

PgUp/PgDn select

- See 'Zoom out'

Mouseclick

- Check if clicking on a hubsection-title makes the semantic zoom zoom out.
- Check if the clicked hubsection is positioned correctly (on the left side of the screen).

Content

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

Functionality

- Check if correct hubsections with correct titles are displayed when zoomed out

Overlay

Content

Create

- Check if adding data in the overlay and clicking the execute button makes the data appear in the application.

Read

- Check if data in the controls is correct

Update

- Check if editing data in the overlay and clicking execute makes the data change in the application

Delete

- Check if removing data in the overlay and clicking execute makes the data dissapear in the application.

Navigations

- Check if we navigated to the right page when control is activated.

Functionality

Open overlay button

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

Close overlay button

- Check if using the button intended to close the overlay, triggers the overlay appearance

Close overlay with “esc”

- Check if overlay closes when pressing the “esc”-key

Control functionality

- Check if “execute” button only highlights when correct datafields are filled in

Ordering

- Check if Combobox ordering function orders the page correctly

Functionality

Scrolling

- Check “Mousewheel” scroll functionality
- Check “Click scrollbar & drag” functionality
- Check “Click scrollbar arrows” functionality
- Check if hovering over the scrollbar makes the scrollbar highlight
- Check if hovering over the scrollbar arrows makes the scrollbar arrows highlight

Buttons

Toggle Button

- Check Toggle Button initial toggle state
- Check if Toggle Button toggle-state is changed when clicked
- Check if Toggle Button is highlighted when hovering over the element
- Check accessibility of Toggle Button with ‘tab’ & “arrow” keys
- Check if Toggle Button responds to ‘space’ key
- Check if hovering over Button makes “extra info”-field appear above it

Button

- Check if Button is highlighted when hovering over the element
- Check if Button executes correct action when clicked
- Check accessibility of Button with ‘tab’ & “arrow” keys
- Check if Button executes correct action when hitting enter (if selected)
- Check if Button executes correct action when hitting space (if selected)
- Check if hovering over Button makes “extra info”-field appear above it

ListItems

- Check if ListItem executes correct action when clicked
- Check accessibility of list with ‘tab’ & “arrow” keys
- Check if ListItem executes correct action when hitting enter (if selected)

- Check if ListItem executes correct action when hitting space (if selected)
- Check if ListItem is highlighted when hovering over the element

Edits

TextBoxes

- Check accessibility of Textbox with “tab”-key
- Check accessibility of Textbox with mouseclick
- Check dataverification
  - A red sentence will appear if the data input from the user does not follow the correct data guidelines
- Check if clicking the “x” in the Textbox sets it back into default state
- Check if Textbox is colored gray when no data is filled inCheck the by default selected textboxCheck if the default Textbox value is visible when not used.Check clipboard functions with right mouse click:
  - Cut
  - Copy
  - Paste
  - Undo
  - Select all
- Check if typing text with the keyboard when Textbox is selected, results in the text beïng added into the Textbox
- Check if hovering over the Textbox makes the mouse change into a line
- Check if hovering over the Textbox makes the Textbox highlight
- Check if selecting the Textbox makes the Textbox highlight even more and makes a flashing line appear inside of it

Searchbox

- Check accessibility of Searchbox with “tab”-key
- Check accessibility of Searchbox with mouseclick
- Check the searchboxes default state: fully gray
- Check the Searchboxes default text
- Check if the Searchbox highlights when hovering over it
- Check if the Searchbox highlights even more when selecting it
- Check clipboard functions with right mouse click:
  - Cut
  - Copy
  - Paste
  - Undo
  - Select all
- Check if typing text with the keyboard when Searchbox is selected, results in the text beïng added into the Searchbox
- Check if typing randomly, without selecting anything, makes your text appear in the searchbox

- Check if hovering over the Searchbox makes the mouse change into a line
- Check if pressing escape makes the searchbox empty
- Check the searchfunction for pressing “Enter”
- Check the searchfunction for clicking the magnifying glass
- Check if the magnifying glass highlights when hovering over it
- Check if selecting the Textbox makes the Textbox highlight even more and makes a flashing line appear inside of it

Combobox

- Check if ComboBox is highlighted when hovering over the element (only when in default modus)
- Check if ComboBox opens when clicking it
- Check accessibility of ComboBox with ‘tab’ key
- Check if combobox opens with the “enter” key
- Check if combobox opens with the “space” key
- Check combobox scrolling functions
- Check if:
  - ComboBox changes the selected item into the item that is clicked
  - This change responds with the proper action/ordering
- Check if ComboBox responds to ‘arrow’ key
- Check if ComboBox responds to ‘tab’ key
- Check if TextBox responds to ‘enter’ key
- Check all highlight functions

Custom

- Things we may find in the future that dont have a specific place within the checklist

Testing: How to

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.

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 steps as 'adding a Coded UI Test project'
- 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 I named my UI Map “UIMap_ClinicHubPage”, this is what I would have to add:

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

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

```
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. Dont 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 higlighted within the application with a blue borded 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 hiërarchy 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 hiërarchy detection. To properly recognize the correct hiërarchy, 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 testprogram itself, we sometimes gave AutomationId’s 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: I gave a unique Id to a list, so I can later easily access the childs of that list

I wanted to give a unique Id to the list of Studies-searchresults, so I searched the XAML file for a while, tried naming some different grid's and gridviews, untill I named the right one. Now if I select that list with the Coded UI Test Builder, the name I gave to it will appear as AutomationId, as seen below.

Testmethods

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 sepcific, 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 set up like this:

```
[TestMethod]
public void METHODNAME()
{
  //Implementation and assertion of TestMethod here
}
```

The '[TestInitialize]' tag

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

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

The '[TestCleanup]' tag

This will be executed *after every* '[TestMethod]'. The same as with the TestInitialize, only after each and every testmethod. 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, bacause we dont want to reload all the data every few seconds for the UI tests.

The ‘[DataSource()]’-tag

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

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

Datasource uses 4 variables to connect to a datasource:

- Type of dataconnection (ex. CSV, XML,...)
- Path to datafile
- Table name
- Access method (sequential/random)

If you add a datatable to a testmethod, the method will be executed for every row in the table. The first row will be ignored and can be used to asign variable names to your data for easy accesability.

Testcontext

“TestContext”, which is a property declared at the bottom of each testproject, 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

In some testproject, 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.

Ex.

Generating controls

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

- 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 UIMap.
- 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 Textfont (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 proporties (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.UICalidosMaatWindowsWindow.UIHomeTitleText;
```

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 can make an assertion for a specific property, by selecting it and clicking “Add Assertion” in the top 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 below.

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

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 above, is the UI Map of the testproject, 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 testmethods or within the partial UI Map class (The part that doesnt contains auto-generated code). Changing the expected value can be done like below:

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

BaseClassCodedUI

This is a baseclass 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 testscenario's (that don't require page-specific control's but overal control's, available on every page) to this class.

The BaseClassCodedUI is a partial class. Which means it is divided into multiple subclasses that alltogether make the full class.

The partial classes are named in a way it seemed most logical to us. We divided them into (currently) 5 partial classes:

- BaseClassCodedUI.cs
  - Contains methods that are a base for other methods within the 'baseclass'
- BaseClassCodedUI.Content.cs

    - Contains methods that involve content verification
- BaseClassCodedUI.Navigations.cs
    - Contains methods that involve navigations
- BaseClassCodedUI.States.cs
    - Contains methods that involve the handling of different states
- BaseClassCodedUI.Functionality.cs
    - Contains methods that involve interaction with controls

Every testproject that is created, has to inherit from this baseclass.

Adding Coded UI Tests to Build process

Some information considering this part can be found on below URL. This MSDN link has not yet been updated to Visual Studio 2015, but will generally be the same process to set up.

URL: https://msdn.microsoft.com/en-us/library/ms182465%28v=vs.110%29.aspx#Anchor_3

Extra informatie

Bijscholingen

Does Not Apply

Nieuwe contacten

Does Not Apply

Literatuur

- 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_prev iew/
- 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://blogs.msdn.microsoft.com/gautamg/2010/01/05/3-introducing-sample-excel-extension/
- http://blogs.msdn.com/b/dpksinghal/archive/2011/09/28/how-to-test-deep-hierarchy-controls-usi

ng-coded-ui-test-in-wpf.aspx

- http://blogs.msdn.com/b/tapas_sahoos_blog/archive/2011/11/07/troubleshooting-record-and-playback-issues-in-coded-ui-test.aspx
- https://blogs.msdn.microsoft.com/tapas_sahoos_blog/2010/12/27/decoding-the-coded-ui-test-playback-failure-search-may-have-failed-at-controlx-as-it-may-have-virtualized-children/
- http://www.codeproject.com/Articles/172391/UIAutomation-Coded-UI-Tests-AutomationPeer-and-WPF
- http://stackoverflow.com/questions/10699795/vs-team-test-multiple-test-initialize-methods-in-test-class