Jani Luostarinen

Web Application Test Automation with Robot Framework

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communications Technology

Thesis

7 May 2015

Tekijä Otsikko Sivumäärä Aika	Jani Luostarinen Web-sovellusten testausautomaatio Robot frameworkilla 41 sivua + 8 liitettä 7.5.2015
Tutkinto	insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja	lehtori Pasi Ranne

Insinöörityön tavoitteena oli tutkia Robot frameworkilla rakennetun testausautomaation kannattavuutta ja sen vaikutusta projektin testausvaiheen fyysiseen työmäärään. Lopputulemana syntyi helposti laajennettava testauskonfiguraatiokokoelma, joka voitaisiin ottaa käyttöön vain muutamalla muutoksella riippuen projektista.

Ensin testausautomaation teoriaa tutkittiin, jotta voitaisiin arvioida sen hyötyjä ja ongelmia. Robot framework valittiin testausautomaatio viitekehykseksi, koska se on avoimen lähdekoodin järjestelmä ja täten ilmainen. Robot framework on myös alustariippumaton, joten se on hyvä valinta virtuaaliseen järjestelmään.

Tuloksena saatiin konfiguroitava kokoelma testidatatiedostoja, joita voidaan helposti suorittaa Robot frameworkilla. Insinöörityö käsittelee käytettyjen tekniikoiden teoriaa, ja siihen on dokumentoitu toimivan testausympäristön rakennusvaiheet. Se tarjoaa myös pohjan toimivalle testausautomaatiokokoelmalle.

Riippuen projektista ja sen käyttötapauksista, testauskokoelma vaatii muutamia muutoksia ennen käyttöönottoa. Kokoelma itsessään tarjoaa perustan mille tahansa testausautomaatioprojektille, jonka testauksen kohteena on Service Now IT -palvelunhallintajärjestelmä.

Avainsanat	testausautomaatio, avainsana pohjainen, web-sovellus

Author Title Number of Pages Date	Jani Luostarinen Web Application Test Automation with Robot Framework 41 pages + 8 appendices 7 May 2015
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructor	Pasi Ranne, Senior Lecturer

The goal of this thesis was to provide a proof of concept that keyword-driven test automation with the Robot framework is feasible and that it would actually reduce the amount of manual work during the testing phase of projects. The result was an easily expandable demonstration configuration (test data suite), which could be taken into use with a few customizations depending on the project.

First, theory of test automation was examined to assess the benefits and the issues in test automation. The Robot framework was chosen as the test automation framework, because it is open source, thus making it free to use, and because it is platform independent, thus a great choice of software to be run on top of a virtual system. Automated tests were run against Service Now's Service Catalog homepage. The homepage is a web application similar to an online shop. It consists of order forms built for an organization's needs to support their internal order processes. These processes can include, for example, ordering a new laptop for an employee. The test automation, when executed, navigates in the webpage, fills in the fields of the order forms, submits orders, reports possible errors and provides statistics. As a result, the test cases were written according to their equivalent use cases.

The result was a configurable test data suite: a set of test data files, which can be easily deployed for the Robot framework to execute. The thesis provides theory of the used techniques and instructions on how to implement a working test environment and the basis for a working test automation suite.

Before taken into use, the suite requires a few customizations that are needed depending on the project and the essential use cases. The suite can be used as a base for any Service Now platform test automation project. From a technical point-of-view, the goals of this thesis were reached, but studying the advantages and disadvantages of acceptance test-driven development could have been emphasized. Also more comparison between a human tester and a computer tester could have been made, to provide an easier choice for those who are contemplating the utilization of test automation in software development projects.

Keywords	test automation, keyword-driven, web application

Contents

List of Abbreviations

SUT	*System under test*: a specific target system, which test automation is run against.
ATDD	*Acceptance test-driven development*: a methodology which provides a common language between the end users, the developers and the testers.
JVM	*Java Virtual Machine*: a process that executes a computer program compiled into Java bytecode. Distributed together with Java Runtime Environment (JRE) by The Oracle Corporation.
.NET	*Dot net software framework*: a software environment that executes programs written for .NET Framework. Developed by Microsoft and used for creating software for Windows based platforms.
TSV	*Tab-separated values*: a simple text file format for storing data in a tabular structure.
reST	*reStructuredText*: a textual data file format used in the Python community for technical documentation.
API	*Application programming interface*: an API provides a set of routines, protocols and tools for easier development of programs. Usually a library that specifies object classes, variables, data structures and other routines.
HTML	*HyperText Markup Language*: a standard markup language used to create web pages. Written in the form of elements, which consist of tags. Designed to display data.

XML	*EXtensible Markup Language*: a standard markup language used to describe data. Written in the form of elements, which consist of self-defined tags.
PaaS	*Platform as a Service*: a set of software applications. A distribution model made available to users usually over the Internet. Applications are hosted by a separate service provider.
ITSM	*IT service management*: an integrated, process based set of best practices which helps manage IT services.
UI	*User interface*: an important part of a program, which enables users to interact with the software.
VM	*Virtual Machine*: an emulation of a computer system, which includes all components of a real computer system such as hardware and software.
GUI	*Graphical user interface*: a part of a program, which enables users to interact with the software graphically through icons and visual indicators.
LXDE	*Lightweight X11 Desktop Environment*: a fast and energy-saving desktop environment available for Linux based distributions.
APT	*Advanced Package Tool*: a free software user interface that handles software package installations and removals on various Linux distributions.
X11	*X Window System*: a windowing system for bitmap displays, provides a framework for graphical user interfaces.
CLI	*Command-line interface*: enables users to interact with a program through commands in the form of text.

URL	*Uniform resource identifier*: a term for describing all target addresses and names, which refer to objects in the Internet.
stdout	*Standard output*: a file handle for processes to print normal information in the Command line interface.
id	*Identifier*: a name, which is used to identify an object or a unique class of objects. An identifier can be a word, number, letter, symbol or a combination of them.

1 Introduction

As modern applications and software develop, become larger and more complex and at the same time are divided between multiple platforms, the importance of software quality and usability become more and more important. Nowadays millions of different applications are easily available for consumers through a wide variety of sources and for various devices. The pressure of developing functional software fast is ever growing, so why not utilize fast computers to remove some of the already limited amount of time reserved for projects and especially for testing? Test automation might provide a solution, because the calculation power and efficiency of a computer is used to support developers to make better software.

This final year project was made for Symfoni Finland Oy for studying the use of test automation in Service Now Service Catalog projects. The Service Catalog is a user-friendly webpage for ordering goods and services through Service Now and these goods and services are modeled with web order forms also known as Catalog Items. The Service Catalog is used by organizations to support their internal ordering processes. These ordering processes can help, for example, in the enrollment of a new employee. The new employee can visit the page and order the needed tools, for example a mobile phone, a laptop or something else for the job in question. One specific Service Catalog project for a customer company started an idea which needed more clarification. In that project, the Service Catalog included multiple, complex and large-scale Catalog Items. As testing all these order forms, their client-side and supporting server-side functionalities would require a lot of manual work, the requirements were to reduce the amount of actual work hours during a testing phase and to provide a packaged solution, which a customer company could utilize by making only a few changes. The size of the Catalog in the previously mentioned project is quite average, but the complexity of the forms makes it a tedious job to test manually. Building a test automation suite might also benefit existing or future Service Catalog projects, which can include hundreds of Catalog Items. Also generic parts of the test automation could be used in other Service Now implementation projects in general.

The purpose of this final year project was to provide a proof of concept that test automation could be used in customer projects and to study the benefits of acceptance test-driven development (ATDD) using the Robot framework test automation framework for the Service Now service management platform. More specifically, the testing automation was run using the Selenium2Library and Selenium 2.0 WebDriver against the Service Catalog homepage of Service Now and different sub-sections of the homepage. Additional custom keywords were used to support test cases for identifying Service Now web elements. As the Robot framework was installed on a virtual machine running a Linux distribution, the available web browsers did not include Internet Explorer. Selenium 2.0 uses and directly supports the latest version of Mozilla Firefox web browser; thus it was used as the main client.

This thesis covers theory regarding the technologies used in the project and introduces the used software tools. It also describes setting up a testing environment, testing a web application and graphical user interface elements in practice and testing results analysis.

2 Test Automation in General

Test automation is carried out with a use of special software to control the execution of predefined test cases. The testing results (outcomes) are then compared to the expected outcomes. Automation helps in testing repetitive tasks, which would be difficult, or would require too much time and resources (manpower) to be run manually. [1, 74.]

Test automation is an attractive concept, but achieving goals might require a lot of work. Keeping this in mind should help in implementing automation well. A test automation project with a wide scope should be considered as a separate software project [2, 2]. Because of these reasons, the scope of the practical testing in this thesis is kept quite narrow and specific to certain areas of the system under test (SUT).

2.1 Benefits of Test Automation

Although implementing sufficient test automation is time consuming, it has a few benefits, when compared to human testers. Performing repetitive tests might be boring to a human tester, which might affect the accuracy of testing. When using test automation, testers can concentrate on more demanding and rewarding work. Thus better results can be accomplished. [2, 3.]

Test automation is much faster than human testing. After test cases are implemented, the automation can accomplish hundreds or even thousands of tests in a matter of minutes. The same amount of tests might take days from a human tester.

One-time setup is a clear benefit in test automation. After the implementation of a test suite, it is possible to run the test suite multiple times, and also it is re-usable. When a new version of a system is taken into use, the existing test automation can be used for regression testing. [2, 3.]

Obviously one of the main benefits of test automation is the amount of manual labor during the testing phase. The amount of manual work hours can be extremely small, compared to the time taken by manual testing.

2.2 Possible Caveats of Test Automation

Everything has negative aspects and this is the case with test automation as well. Defining use cases and matching test cases against them and to define expected outcomes is problematic for many organizations. It might take more time to change an organization's processes in testing than implementing the test automation.

Analyzing the test outcomes is also important and if the analysis is incorrect, the benefits of test automation are tenuous. The test case might fail, but that does not mean there was a bug in the SUT. The reason might have been, all along, that the test case was written poorly.

The main caveat in test automation is the fact that building it takes a lot of time. Depending on the scope of the test automation, the implementation of the test automation can vary from days to weeks, even months. It is a slow process to, for example, investigate the web elements of a web page and then figure out the methods for invoking the web elements. Especially GUI testing automation is difficult, because UI objects are hard to recognize. GUIs are also updated and changed often, which causes problems, because test automation needs to be updated accordingly. [2, 4.]

3 Robot Framework

The Robot framework is an open source test automation framework, which is used for acceptance testing and acceptance test-driven development (ATDD). It has a tabular test data syntax and it uses the keyword-driven testing approach. Because the Robot framework follows a modular architecture, the testing capability of it can be extended by test libraries programmed with Python or Java. [3.]

It is considered to be a third generation test automation framework. Configuring the Robot framework test data does not necessarily require programming or scripting, but just altering data. The concept of keyword-driven testing makes it easier to create test data driving the test execution. [2, 11.]

The core framework of the Robot framework is implemented with Python, thus making it operating system and application independent. It also runs on Jython (JVM) and IronPython (.NET). The possibility of running it on any operating system, which supports Python, makes it highly flexible. [3.]

The Robot framework software is released under Apache License 2.0, and also most of the libraries and tools in the ecosystem are open source. Nokia Networks supports the development of the core framework. [3.]

The Robot framework processes the test data when it is started. The framework does not need to know the target system, but utilizes test libraries to interact with it. The libraries use application interfaces or separate test tools as drivers. [4.]

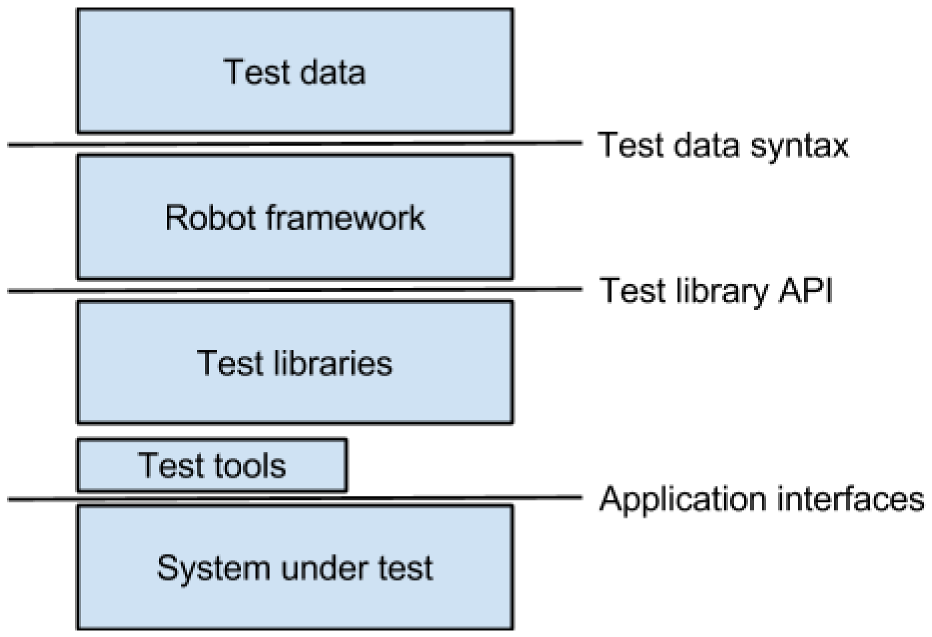


Figure 1. High level architecture. Modified from robotframework.org. [3.]

As figure 1 illustrates the actual framework relies on the created test data and then utilizes the data with the help of different libraries. The libraries then use tools and interfaces to command the system under test to perform various functions.

3.1 Acceptance Test-Driven Development

Acceptance test-driven development (ATDD) is a methodology, which provides a common language between the end (customer) users, the developers and the testers. It is a process, which helps the developers and testers to understand the customer's requirements. [6, 5.]

ATDD covers acceptance testing, but differs from it, requiring use cases and acceptance tests before any development. The emphasis in ATDD is on communication between the user, developer and tester.

3.2 Tabular Test Data Syntax

As seen in listing 1, Robot framework relies on configuration files which follow a specific intended document formatting. This format is called tabular test data syntax. This means that after the beginning of a section every new line of text needs to be intended with two or more spaces. Test data is defined in configuration files, using a specific syntax. Multiple test cases in a file creates a testing suite and multiple files in a directory creates a nested structure of test suites. [3.]

```
*** Settings ***
Documentation     A test suite with a single test for
valid login.
...
...               This test has a workflow that is
created using keywords in
...               the imported resource file.
Resource          resource.txt

*** Test Cases ***
Valid Login
    Open Browser To Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

Listing 1. Example test data file. Reprinted from robotframework.org. [6.]

The Robot framework supports four different file formats. The tabular format can be defined using HyperText Markup Language (HTML), tab-separated values (TSV), plain text or reStructuredText (reST). According to the Robot framework user guide, plain text file format is recommended. Listing 1 describes an example of a plain text test data file. When defining plain text test data files, they should have a file extension of “.txt” or “.robot”. [4.]

3.3 Keywords and Syntax in More Detail

The framework starts the execution with parsing the test data files. Next it uses keywords defined in the test data files and starts to interact with the target system. The libraries communicate with the system directly, but also other testing tools can be utilized as drivers. [3.]

As listing 2 illustrates, the keywords act as commands for the framework to know what should be executed on the target system. It is possible to utilize predefined keywords found in existing libraries, or new keywords can be defined to new libraries.

```
*** Settings ***
Library             Selenium2Library

*** Variables ***
${SERVER}           localhost:7272
${BROWSER}          Firefox
${DELAY}            0
${VALID USER}       demo
${VALID PASSWORD}      mode
${LOGIN URL}        http://${SERVER}/
${WELCOME URL}      http://${SERVER}/welcome.html
${ERROR URL}        http://${SERVER}/error.html

*** Keywords ***
Open Browser To Login Page
    Open Browser    ${LOGIN URL}    ${BROWSER}
    Maximize Browser Window
    Set Selenium Speed    ${DELAY}
    Login Page Should Be Open
```

Listing 2. Example test data file using keywords. Reprinted from robotframework.org. [6.]

Different sections (test data tables) of a test data file are separated by a line of text starting and ending with one or more asterisk (*) characters. Also sections should be separated with a line break, to keep the data human-readable and neat. Global variables are initialized in the Variables section. A variable starts with a dollar sign ($) and the name can be anything as long as it describes its purpose and surrounded by a pair of braces ({}). When defining keywords the first "heading" line should be left unindented. Every word should start with an upper case letter and be separated with a space. When going deeper into the keyword definition, the indentation should be at least two space characters. In listing 2 there are four space characters in the indentation. It is recommended to keep the width of the indentation consistent through all test data files in a test suite. Especially in plain text format, the use of space characters is recommended instead of a tabular character. In addition when referring to a variable in between keywords, at least two space characters should be used before the variable for the Robot framework parser to separate the variables from usual keywords. [4.]

3.4 Selenium2Library

Selenium2Library is a testing library used by the Robot framework. It utilizes Selenium 2 and WebDriver libraries to control a web browser and run tests in a real browser session. It supports most modern browsers. The Mozilla Firefox browser is supported out-of-the-box. Other browsers require an additional driver and setup for Selenium to work with them. The Selenium2Library provides a wide variety of general keywords to support web testing. For the Robot framework to utilize the Selenium2Library, it needs to be imported into the test suite using the “Library” keyword in the Settings section of a suite. [7.]

3.5 Selenium 2 (WebDriver)

Selenium 2 is the latest implementation and part of Selenium’s kit of software tools. It is a test automation tool, which provides an object oriented API. It includes and supports technologies from the WebDriver API and Selenium 1. [8.]

Selenium 2 includes a huge amount of methods for interacting with web elements. Also the project’s documentation is comprehensive and available for everyone on the Internet.

3.6 Report and Log

At the end of executing the test suite files, the Robot framework will provide results debriefing. Result report and log are displayed in HTML format as well as in XML outputs. The report provides statistics, for example pass/fail ratios and run times. The log provides a more detailed view about the test execution. The logs, for example, display separately which keyword was passed and in what keyword the failure occurred. [6.]

4 Service Now

Service Now Inc. was founded in 2003 by Fred Luddy, who was the previous CTO of Peregrine Systems and Remedy Corporation. The company is a provider of Service Management software called Service Now, which as a product, competes with BMC, Computer Associates, IBM and Hewlett-Packard in the area of ITSM applications.

Service Now is a cloud-based platform, which provides multiple ways to automate a company's IT functions. The main focus of the product is to automate and standardize business processes and to provide a single system, which can accommodate different parts of the enterprise areas, such as improving efficiency and lowering operational costs. Nowadays Service Now covers the automation of other, non-IT functions, for instance marketing, facilities and human resources, to list a few.

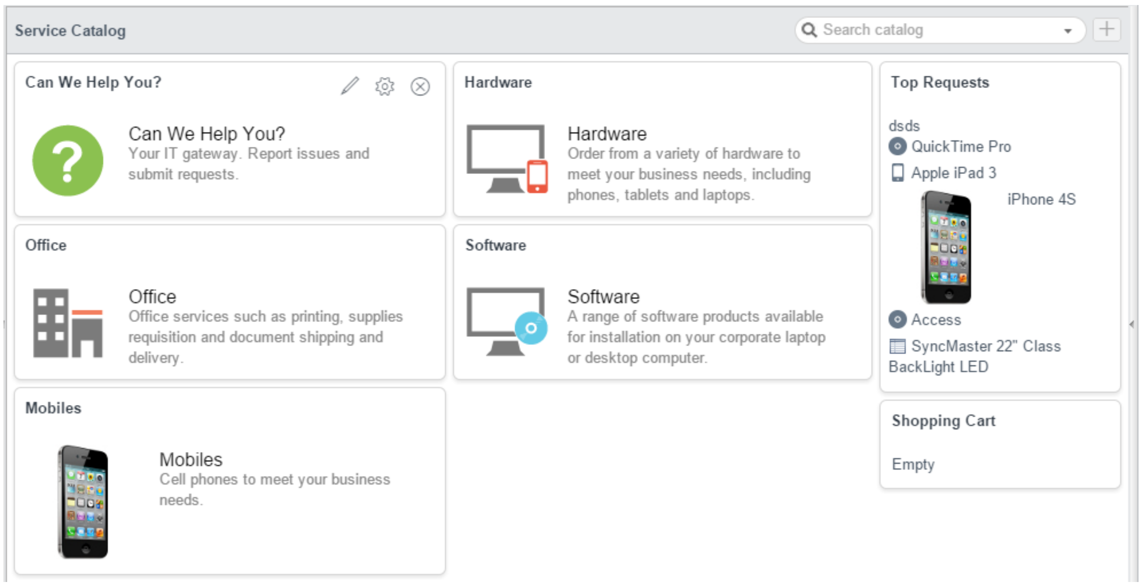
The platform can be roughly divided into five main categories, which are Asset and Configuration, Planning and Policy, IT Service, Non-IT Service and IT Operations management.

The Service Now platform-as-a-service (PaaS) is a complete suite of applications, which are built to be modular. Modularity in Service Now grants a way for customers to pick the applications of their specific need. This means that a company can also activate new applications later on. The platform itself is highly customizable and the look-and-feel can be customized to match an organization's brand.

The test automation is run against the front end of Service Now's Service Catalog application called Service Catalog homepage. The homepage provides a web page from which users can order services or goods, using user-friendly web forms. To support the ordering process, the homepage uses functionalities similar to an "online shop" such as categorized items, a shopping cart and an order summary check out page. [9.]

4.1 Service Catalog Homepage

The Service Catalog homepage is the primary front end for ordering items via Service Now. It is a way for employees to use pre-defined web forms to order goods and services from an IT organization or other departments. Figure 2 displays the Catalog as seen by the user from an actual environment. [8.]



Service Catalog

Search catalog

Can We Help You?

Can We Help You?
Your IT gateway. Report issues and submit requests.

Hardware

Hardware
Order from a variety of hardware to meet your business needs, including phones, tablets and laptops.

Top Requests

dsds
QuickTime Pro
Apple iPad 3
iPhone 4S
Access
SyncMaster 22" Class BackLight LED

Office

Office
Office services such as printing, supplies requisition and document shipping and delivery.

Software

Software
A range of software products available for installation on your corporate laptop or desktop computer.

Mobiles

Mobiles
Cell phones to meet your business needs.

Shopping Cart

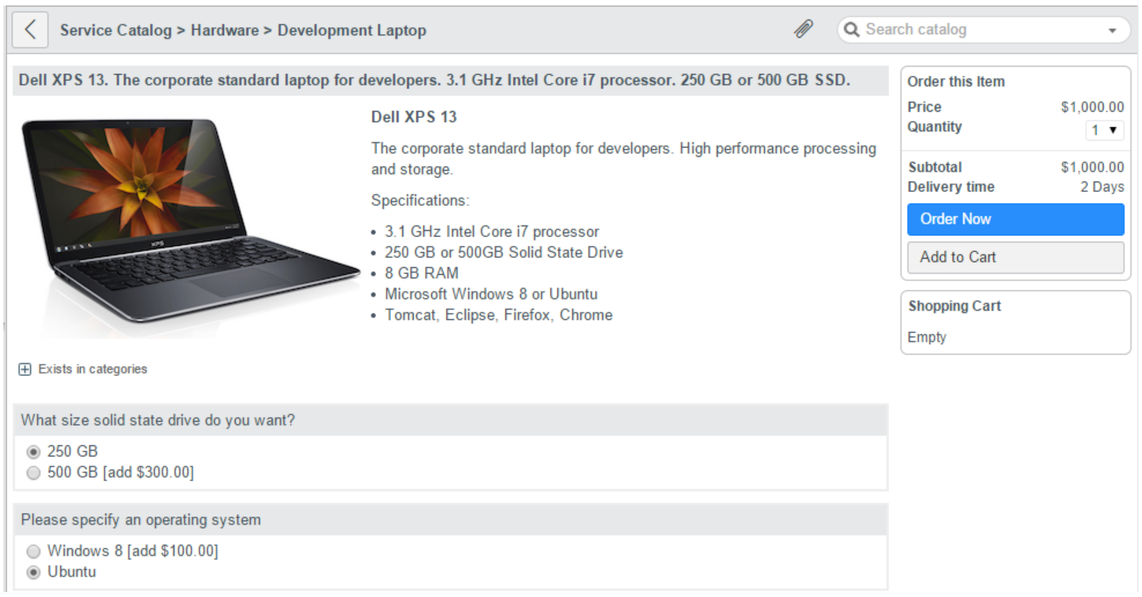
Empty

Figure 2. Service Catalog in Service Now. Screenshot [15].

The Catalog is a categorized listing of Catalog Items, which are provided to end users. For example “Hardware” is a category, which may contain one or multiple Catalog Items.

4.2 Catalog Items

The Catalog Items form the core of the Service Catalog. As seen in figure 3, a Catalog Item is a user-friendly web form, which describes an orderable entity, such as a laptop, a mobile phone, a software for a computer, or even a new stack of business cards. [9.]


Service Catalog > Hardware > Development Laptop

Dell XPS 13. The corporate standard laptop for developers. 3.1 GHz Intel Core i7 processor. 250 GB or 500 GB SSD.

Dell XPS 13

The corporate standard laptop for developers. High performance processing and storage.

Specifications:

- 3.1 GHz Intel Core i7 processor
- 250 GB or 500GB Solid State Drive
- 8 GB RAM
- Microsoft Windows 8 or Ubuntu
- Tomcat, Eclipse, Firefox, Chrome

Exists in categories

What size solid state drive do you want?

[✓ 250 GB]
500 GB [add $300.00]

Please specify an operating system

Windows 8 [add $100.00]
[✓ Ubuntu]

Order this Item

Price	$1,000.00
Quantity	1
Subtotal	$1,000.00
Delivery time	2 Days

Shopping Cart

Empty

Figure 3. A Catalog Item with a shopping cart. Screenshot [15].

A Catalog Item from a technical perspective is constructed from three main elements: Catalog variables, Catalog client scripts and Catalog UI policies. It is possible to define prices to the Items itself or a price for a specific selection. Also, adding a picture and a descriptive text for the item is possible.

4.2.1 Catalog Variables

Catalog variables are input elements, which capture and pass on information about choices the end user makes during an order. For example, a Catalog Item for ordering a new desktop PC can include a variable called “Hard-drive”. This variable could contain choices of different hard-drive sizes. The selection of a hard-drive would then reflect the price of the orderable item, depending on the selected size. The Variables are so-called building blocks for implementing Catalog Items. They provide an easy way to build a web form. [10.]

Service Now provides a set of Variables types:

- Break — a horizontal line across the form
- CheckBox — a checkbox that can be selected or cleared (true/false)
- Containers — a variable that contains other variables for additional layout options

- Date — a date input variable with a calendar widget
- Date/Time — a date and time input variable with a calendar widget
- HTML — user input, or reusable content with a possibility of a simple HTML editor
- Label — a label that is displayed across the form
- List Collector — a list collector interface for selecting multiple records from a table
- Lookup Multiple Choice — creates radio buttons depending on an underlying table it refers to
- Lookup Select Box — creates a choice list (drop down list) depending on an underlying table it refers to
- Macro — inserts an UI macro (a modular, dynamic, reusable component that is constructed with Jelly scripting)
- Macro with Label — the same as Macro, but with a label element
- Masked — a text input box that masks out the entered text with an asterisk symbol for every keystroke, used for sensitive data such as passwords, can utilize encryption
- Multi Line Text — a multiple-line text input
- Multiple Choice — creates radio buttons for static question choices
- Numeric Scale — creates a horizontal set of radio buttons with numeric options
- Reference — a text input field that references a record on another table
- Select Box — creates a choice list (drop down list) for static question choices
- Single Line Text — a single-line text input field
- UI Page — similar to UI macro, a UI page can contain multiple UI macros
- Wide Single Line Text — a wide single-line text input field
- Yes/No — creates a choice list with Yes and No options

[10.]

4.2.2 Catalog Client Scripts

Catalog client scripts add dynamic effects and validation to forms. Three main types of Catalog client scripts are usually used for Catalog Items. Table 1 describes the types onLoad, onChange and onSubmit, and their run sequence.

Table 1. Catalog Client Scripts. Modified from wiki.service-now.com. [11.]

Script type	When it runs
onLoad()	When system loads a form
onChange()	A field value on the form changes
onSubmit()	A user saves/submits a form

These scripts are shipped to the client-side and executed on the browser. Client scripts in Service Now are implemented with JavaScript and make use of GlideForm API, which provides a wide variety of methods to:

- Get or set Variable values
- Hide or display Variables
- Make Variables mandatory or non-mandatory (enforce user input)
- Validate form submission
- Validate user input

[12.]

GlideForm.js is a JavaScript class used to customize forms, and g_form is a global object in the GlideForm class. [13.]

4.2.3 Catalog UI Policies

UI policies are similar in functionality to client scripts for dynamically changing information on a form. The main difference to client scripts is that no scripting is required to define a UI policy. UI policies are built-in functionalities, which make basic form modifications easier. They follow conditions defined in the UI policy record and depending on the evaluation of the condition the UI policy is executed. Sometimes it is not feasible to implement a script to only change the visibility of one field on a form. In cases like these, UI policies are the better and easier option.

5 Test Environment

For testing and demonstration purposes a test environment was built. The implementation was done keeping minimum requirements in mind. Because of this, it did not fully reflect an actual environment. The demonstration environment did not take into consideration for example the network infrastructure, user authentication methods, or any other real-world situations, which might affect parts of the process flow of the test automation.

For running the Robot framework and executing tests which reflect real-world situations, a few prerequisites had to be fulfilled. The most sensible option was to use a virtual environment with a lightweight operating system and just enough resources. For the test environment, a virtual machine running an up-to-date Ubuntu server 14.04.2 operating system was implemented. Oracle VM Virtual Box Manager was used as the virtualization platform.

A fully operational demonstration instance of Service Now ITSM platform was already running and available for browser access. This included an active user account with the correct user rights. In real-world situations, the user rights would depend on the defined use case.

5.1 Implementation

5.1.1 Installation

After installing and updating an Ubuntu server 14.04.2 virtual machine with Oracle VM VirtualBox Manager, a GUI, Lubuntu Desktop was installed, because a web browser is needed to access web applications. Lubuntu was selected as the desktop manager, because it is lightweight and uses the minimal desktop LXDE and a set of light applications with a focus on performance and energy-efficiency, thus making it a great desktop manager for a virtual server system.

The Lubuntu desktop manager was installed with only the core applications as can be seen from the following command.

```
sudo apt-get install --no-install-recommends lubuntu-
desktop
```

Listing 3. Install Lubuntu desktop manager

The “sudo” command executes a program which gives the current user the security privileges of an administration level user (root). The second command, “apt-get”, pilkku calls a package management tool APT. Three parameters are given to the package management command. The first “install” command is a parameter which tells the package management tool to expect a package name and then install it. The second parameter, “--no-install-recommends”, tells it to install only the core applications. The third parameter is the software package name, in this case “lubuntu-desktop”. The next step was to reboot the system for accessing the Lubuntu Desktop manager.

Most modern Linux distributions come pre-installed with a Python interpreter which is required by the Robot framework. The Robot framework does not yet support Python 3, so the required Python version needed to be checked from the command line (CLI), as the command below shows.

```
sudo python –version
```

Listing 4. Check Python version

The command “python” calls the python interpreter and “--version” parameter tells it to print the version number to the standard output (stdout). Ubuntu server 14.04.2 provides Python version 2.7.6 by default.

There are different ways to install Robot framework. The installation can be done by compiling it directly from the source, or by manual installation. It can also be installed using a Windows installer, as a standalone JAR distribution, or with a Python package manager. [4]. For the most straightforward approach, the Pip, Python package manager was selected and installed.

```
cd /mnt
sudo wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py
```

Listing 5. Download and install Pip

The command “cd /mnt” changes the current working directory to a directory called “mnt”. The second command “wget” calls a program which can download files from a given

URL. The third command tells the Python interpreter to run the downloaded python script which installs the Pip package manager. After this, the Robot framework was installed using the Pip.

```
sudo pip install robotframework
```

Listing 6. Install Robot framework

The command “pip” calls the Pip package manager and the parameter “install” tells it to install a software package defined on the second parameter. The Robot framework installation was verified by executing its runner script.

```
sudo pybot --version
```

Listing 7. Check pybot version

The runner script “pybot” with parameter “--version” outputs the framework and interpreter versions. The versions can be seen from the output below.

```
Robot Framework 2.8.7 (Python 2.7.6 on linux2)
```

Listing 8. Version output

For leveraging the Selenium 2 WebDriver libraries, Selenium2Library needed to be installed as well.

```
sudo pip install robotframework-selenium2library
```

Listing 9. Install Selenium2Library

As seen above, the installation command was the same as for the Robot framework installation. The only difference was the package name given in the second parameter of “pip” command. After a successful installation, the configuration of the test data files could be started.

5.1.2 Basic Configuration

The configuration of the Robot framework test data files was started by writing a simple test suite to be able to confirm correct functionality of the installation. The target of the simple test suite was to control (open and close) the browser, open a correct URL, login into an existing Service Now instance and log out of it.

A test suite file can contain four sections: Settings, Variables, Test Cases and Keywords, but for easier maintenance, Variables and keywords were taken into separate files. As shown in appendix 1, the first file is the main test suite file, which contains settings and a list of test cases. The settings section contains the used keyword libraries and user keyword resource files. The list of test cases define the tests, which are run when the suite is executed.

As described in appendix 2, the test file defines the suite's global Variables. These include the target address, a valid user name and a password for that specific user. The file needs to be included in the suite with a Resource keyword.

The suite's base keywords were defined in a user keyword file as shown in appendix 3. With the help of base keywords the execution would accomplish the actual navigation on the target web page. Majority of the used keywords come directly from the Selenium2Library. Without including the Selenium library, the execution of the test suite would fail and stop. These structure and syntax errors are logged, just like normal test errors into HTML and XML output files, but also output into the standard output of the command line. For the simple test suite, two new keywords "Login" and "Logout" were defined. As the test suite is executed quite fast, a few extra "Wait Until" keyword combinations were needed for taking into consideration network latency, browser rendering speed and of course the speed of human sight.

The fourth test data file contains the defined test cases as seen in appendix 4. For the simple test suite, only one test case, "Test Login", was written. The test cases use keywords defined in the user keyword test data file, but can also contain keywords from the Selenium2Library if required.

5.1.3 Basic Testing

Basic testing was executed running the simple suite with the runner script "pybot" as seen below.

```
pybot base-suite-test.txt
```

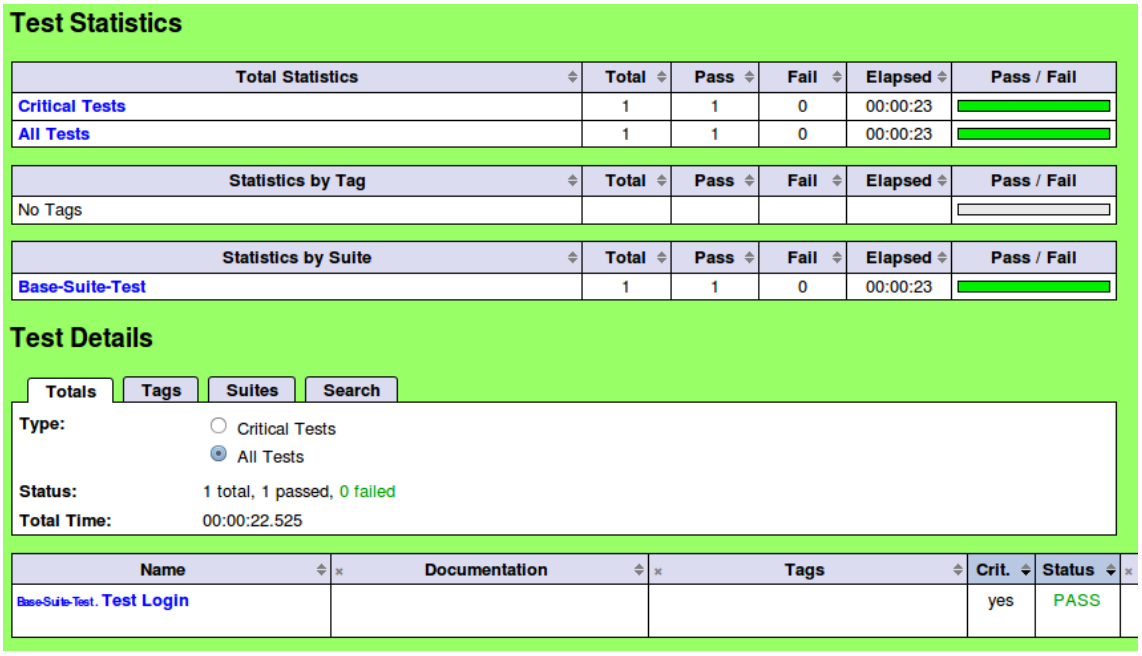
Listing 10. Run test suite

As seen in listing 3, the basic test was successful and the Robot framework created outputs for the test case and for the overall status of the whole suite.

```
=======================================================
========================

Base-Suite-Test

=======================================================
========================

Test Login
Test Login
| PASS |

-------------------------------------------------------
------------------------

Base-Suite-Test
| PASS |

1 critical test, 1 passed, 0 failed

1 test total, 1 passed, 0 failed

=======================================================
========================

Output:  /home/robot-user/Documents/test-
configs/output.xml

Log:     /home/robot-user/Documents/test-
configs/log.html

Report:  /home/robot-user/Documents/test-
configs/report.html
```

Listing 11. Base suite results in Standard output

Also an HTML report was created automatically, which clearly displays statistics of the executed suite as seen in figure 4. Reports should not be too cluttered with information, so for example, a person without technical expertise can also review the results with a simple glance.



Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:23	🟩
All Tests	1	1	0	00:00:23	🟩

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Base-Suite-Test	1	1	0	00:00:23	🟩

Test Details

Totals | Tags | Suites | Search

Type:	○ Critical Tests ● All Tests
Status:	1 total, 1 passed, 0 failed
Total Time:	00:00:22.525

Name	Documentation	Tags	Crit.	Status
Base-Suite-Test. Test Login			yes	PASS

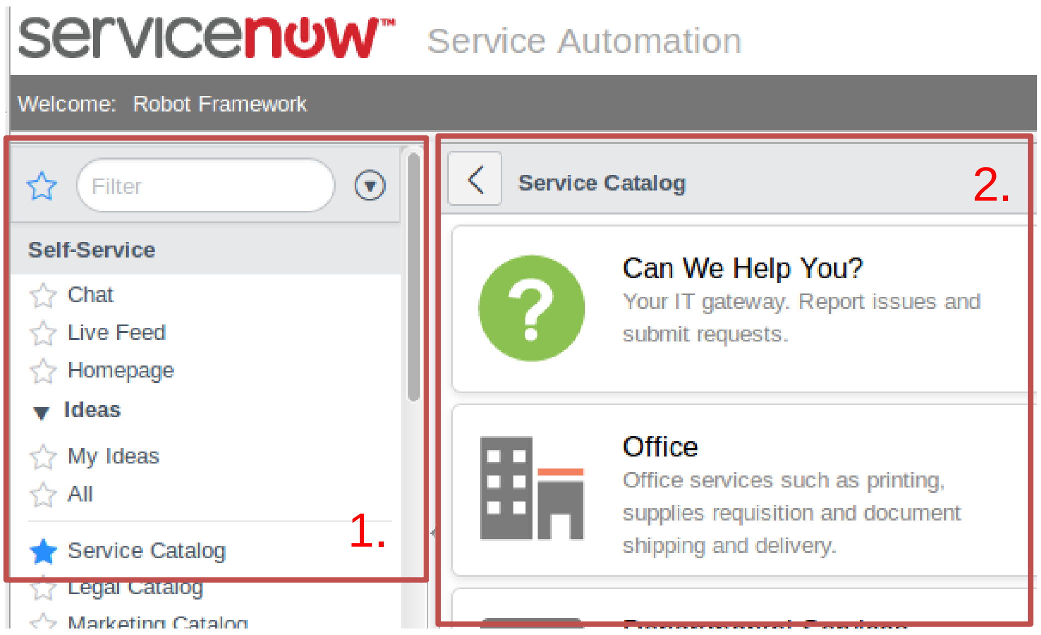
Figure 4. Base suite results in HTML report. Screenshot [16].

The Test Statistics display a “PASS” or “FAIL” tag in the Status column of the test case listing. Color coded values are used for easier visual interpretation. Green color is used for a passed test case, and red color for a failed test case. The statistics also display elapsed run time, start time and end time for each test case, but also for the whole test suite.

6 Configurations for Service Catalog Test Automation

6.1 Common Keywords

The configuration for a specific part of Service Now was started by writing the supposedly generic keywords, which would be needed in, for example, navigating through Service Now's graphical user interface. The existing keywords defined in the base suite were used, as they would provide the ability to login to the instance and close the browser as soon as testing was complete. As seen in figure 5, Service Now's user interface called UI15 is constructed of a few main sections or frames which each contain different functionalities.



Your IT gateway. Report issues and submit requests.

Office services such as printing, supplies requisition and document shipping and delivery.

Figure 5. A part of Service Now's UI15. Screenshot [15].

The main frames of the UI are displayed in figure 5 above. The left most frame numbered with 1 is called Application Navigator, which provides the ability to navigate to applications and modules within Service Now. When a module, in this case, Service Catalog is clicked under the Self-Service application, the content of the module is displayed in the Content Frame, which is numbered 2. The Content Frame is the largest frame and spans from the edge of the Application Navigator, all the way to the right hand side of the browser window. Only a small amount of the Content Frame is shown in figure 5. [14].

The common generic keywords were written on as-need-basis. After a single navigation step such as a button press or clicking a link in the UI, it was decided that the keyword should be a generic one, or a test case specific one. This way, it was possible to expand the resource file gradually and collect only the essential keywords. On the way, a few rules were made up to help future developers to add more generic keywords. These rules were documented in the beginning of the generic resource file as seen in appendix 7.

6.2 Example Use Case

As ATDD states, every use case should be documented before any implementation or any test case. Successfully ordering a sales laptop Catalog Item through the Service Catalog was the first use case and the main flow was defined as follows:

1. Successfully login to Service Now demo instance with the test user credentials
2. Navigate to Service Catalog module under Service Catalog
3. Navigate to Hardware Catalog Category
4. Select “Sales Laptop” Catalog Item
5. Select “Microsoft Powerpoint” CheckBox under “Optional software” Label
6. Input text “Skype pre-installed” into “Additional software requirements” Multi-line Text Variable
7. Order the Catalog Item, by pressing the “Order Now” button on the catalog shopping cart

The post conditions define the outcomes of the use case. There are two possible outcomes, which are success and failure, and these outcomes need to be met by pre-defined conditions. Conditions for a successful outcome were defined as follows:

- Test user (actor) is able to access the demo instance and the Catalog Item via the specified way of navigating
- Test user is able to fill in the required fields of the order form

- Test user is able to submit the order and the order is successfully recorded in the database

The conditions for a failed outcome were defined as follows:

- Test user (actor) cannot access the demo instance, nor the Catalog Item via the specified way of navigating
- Test user receives an error message in any point of the order flow
- An error occurs, which prevents the completion of the order (no error message displayed)
- Test user cannot submit the order, or the order is not successfully recorded in the database

6.3 Example Test Case

The test case definitions are based on the requirements of the use case. The test case was written with keywords, which would match those requirements and fulfill the use case. Listing 4 describes the needed steps (excluding login) for the main flow of the use case. The first line of text is the name of the test case, in this case “Order Sales Laptop”.

In addition to test cases, a suite can have a setup and a teardown. The suite setup is run before any test case and the suite teardown is run after everything else. The “Login/Logout” keywords were not needed for every test case separately, but they could be utilized directly in the settings section of the testing suite as seen in appendix 5. The “Login” keyword was an ideal step for checking an obvious precondition: successful login with an existing user account. The “Logout” keyword on the other hand was the last step in any case, because it is pointless to leave the browser running and taking resources after test execution. Because of this, the first needed keywords were for navigating to the correct module, as line two of listing 4 describes.

```
Order Sales Laptop
  Open Module  Self-Service  Service Catalog
  Select Main
  Click Link  link=Hardware
  Select Main
  Click Link  link=Sales Laptop
  Select Main
  Select Checkbox
ni.IO:3cfe1f290a0a0a6a01ee1623f4982abd
  Input Text  IO:f3776ac9c0a8010a003825c70f35fab7
Skype pre-installed
  Click Button  oi_order_now_button
```

Listing 12. Order Sales Laptop test case

The keywords in line three select and set the focus to the Content frame. “Select Main” could be easily re-used multiple times as it was included in the common keywords’ resource file. Line four tells the web-driver to click a link the title of which is “Hardware” in the Content frame. As the page, which the link is pointing to, loads and the frame refreshes, it is needed to again set the focus to the Content frame (line five). Steps in line four and five are repeated with a different link parameter to drill deeper into the stack. Next the Catalog Item is opened into the frame and the form fields (Catalog Variables) can be filled with a value specified in the use case (lines eight and nine). Service Now creates an “id” parameter automatically for the Variables as they are rendered. Thus the format of the “id” is not very readable. Finally in line ten the order is submitted by pressing the “Order now” button. The full test case configurations can be seen in appendix 8.

7 Testing

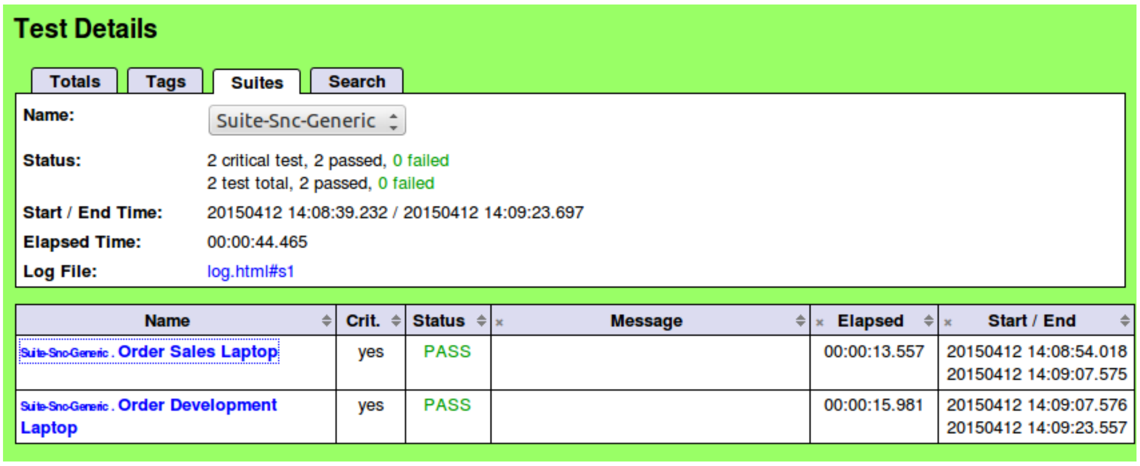
After the needed test cases were written in to the test case file, the suite could be run again, using the “pybot” starter script. The suite makes use of three resource test data files in the same way as in the basic test suite referred to in the earlier chapters. One additional setting was added to the Variables section of the settings file. As seen in appendix 6, the variable “${family}” is used for recognizing the latest three Service Now versions: Dublin, Eureka and Fuji (latest). The Application navigator behaved differently for an older version of Service Now. Thus an if-clause was needed in the common keywords file, displayed in appendix 7, to use a condition before running a keyword, which selects the Application navigator’s frame. Because of the updated GUI, it was decided not to support older versions of Service Now, although there is still some organizations using the older ones.

7.1 Analyzing Successful Tests

For the demonstration, the Catalog Items were known for sure to be completely working and for that reason, the test suite would return only successful results. This would assure that the suite is configured and working as intended.

After running the suite with two test cases, one for ordering the sales laptop as described in chapter 6.3 and a second similar test case as described in appendix 5, the report and log could be examined.

As seen in figure 6, both test cases were successfully completed. In this run, the Order Sales Laptop test case took just over 13 seconds to execute and Order Development Laptop took almost 16 seconds. Ten test suite executions were completed and an average time of 15 seconds per test case was measured. It was also measured that in ten runs a human tester completed each of the test cases with an average time of 30 seconds.



Test Details

Totals | Tags | Suites | Search

Name:	Suite-Snc-Generic
Status:	2 critical test, 2 passed, 0 failed 2 test total, 2 passed, 0 failed
Start / End Time:	20150412 14:08:39.232 / 20150412 14:09:23.697
Elapsed Time:	00:00:44.465
Log File:	log.html#s1

Name	Crit.	Status	Message	Elapsed	Start / End
Suite-Snc-Generic . **Order Sales Laptop**	yes	PASS		00:00:13.557	20150412 14:08:54.018 20150412 14:09:07.575
Suite-Snc-Generic . **Order Development Laptop**	yes	PASS		00:00:15.981	20150412 14:09:07.576 20150412 14:09:23.557

Figure 6. Service Catalog test suite report. Screenshot [16].

It was assumed that the human tester is familiar with the SUT and an experienced tester with a good overall knowledge of the test cases. Even without any spelling mistakes, miss clicks, or other manual errors, it could be seen that a human tester is almost two times slower than the Robot framework when executing test cases. The elapsed time in test cases executed by the Robot framework could be reduced even more by optimizing the underlying hardware and removing unnecessary “wait for something to load” key-words from the test data files.

Of course, if the underlying configurations in Service Catalog changed, the test cases would require a change, or at least a test run to see if they need to be changed. This too takes time and the overall difference between a computer and a human tester’s theoret-ical elapsed times would be smaller. A human tester could adapt to a configuration change much more quickly and continue testing, taking the change into consideration. However if the Robot framework can tirelessly run through as many test cases as given to it, it is safe to assume that it is much faster.

7.2 Analyzing Unsuccessful Tests

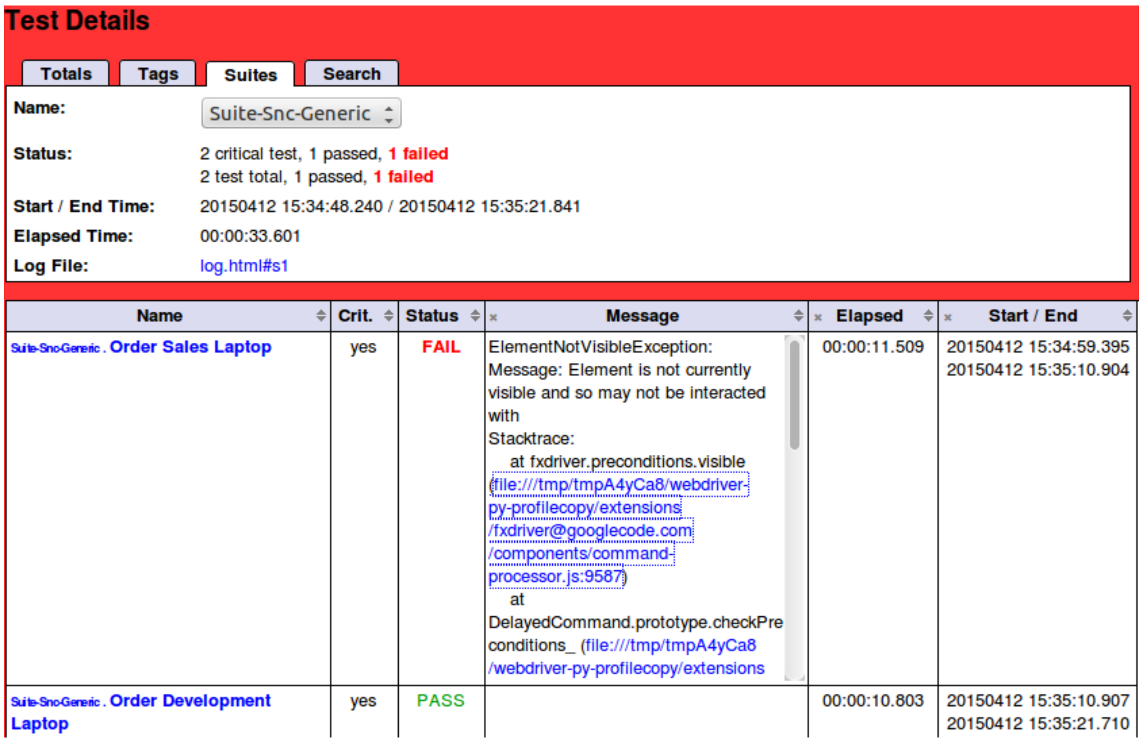
For demonstration purposes an error was induced into the main flow of the sales laptop Catalog Item. As listing 5 shows the error is caused with an onChange Catalog Client Script, which hides the “Additional software requirements” Multi-line text Variable, if “Mi-crosoft PowerPoint” CheckBox Variable is selected.

```
function onChange(control, oldValue, newValue,
isLoading) {
          if (isLoading || newValue == '') {
                    return;
          }
          else {
                    if (newValue == 'true') {

          g_form.setDisplay('add_software', false);
                    }
          }
}
```

Listing 13. An onChange client script

An onChange script is triggered when the related element's, in this case the PowerPoint CheckBox's, value is modified. When the function starts, the first if-clause checks, if the page is still being loaded and, if the changed value is empty. It is unnecessary to run the script during a page load, or if the value does not actually contain anything. The next step is to check that the CheckBox is actually selected. If it is, the Multi-line text field is hidden with a method from the g_form API. The above mentioned Client Script might be an undocumented change to the system or just some other mishap done by an enthusi-astic developer.

As seen in figure 7, a fail tag is visible on the Status column of the Order Sales Laptop test case. The failure is emphasized with a red background color. A clear error message can be found in the Message column that indicates a missing element.

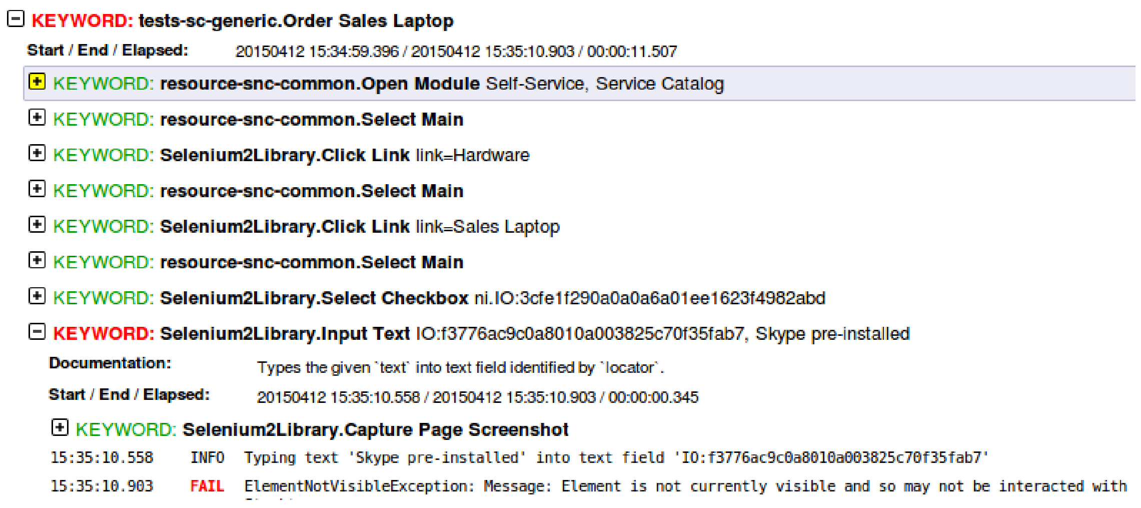


Name:	Suite-Snc-Generic
Status:	2 critical test, 1 passed, 1 failed 2 test total, 1 passed, 1 failed
Start / End Time:	20150412 15:34:48.240 / 20150412 15:35:21.841
Elapsed Time:	00:00:33.601
Log File:	log.html#s1

Name	Crit.	Status	Message	Elapsed	Start / End
Suite-Snc-Generic . Order Sales Laptop	yes	FAIL	ElementNotVisibleException: Message: Element is not currently visible and so may not be interacted with Stacktrace: at fxdriver.preconditions.visible (file:///tmp/tmpA4yCa8/webdriver-py-profilecopy/extensions/fxdriver@googlecode.com/components/command-processor.js:9587) at DelayedCommand.prototype.checkPreconditions_ (file:///tmp/tmpA4yCa8/webdriver-py-profilecopy/extensions	00:00:11.509	20150412 15:34:59.395 20150412 15:35:10.904
Suite-Snc-Generic . Order Development Laptop	yes	PASS		00:00:10.803	20150412 15:35:10.907 20150412 15:35:21.710

Figure 7. Service Catalog test suite report with a failure. Screenshot [16].

The test report in figure 7 is a clear indication of a failed test case. But what actually went wrong? The report gives an overall picture of the status, but does not tell exactly what the reason was for the failure. The missing element could be anywhere on the form. If the form would be larger with a lot of fields and dynamic functionality, it would require time to troubleshoot the issue and find the culprit. This is when the log comes in to the picture. By clicking the test case name, the log file is opened to the browser, also in HTML format. As figure 8 displays, the log file gives more of an in-depth view into the issue.



```
KEYWORD: tests-sc-generic.Order Sales Laptop
  Start / End / Elapsed:    20150412 15:34:59.396 / 20150412 15:35:10.903 / 00:00:11.507
  KEYWORD: resource-snc-common.Open Module Self-Service, Service Catalog
  KEYWORD: resource-snc-common.Select Main
  KEYWORD: Selenium2Library.Click Link link=Hardware
  KEYWORD: resource-snc-common.Select Main
  KEYWORD: Selenium2Library.Click Link link=Sales Laptop
  KEYWORD: resource-snc-common.Select Main
  KEYWORD: Selenium2Library.Select Checkbox ni.IO:3cfe1f290a0a0a6a01ee1623f4982abd
  KEYWORD: Selenium2Library.Input Text IO:f3776ac9c0a8010a003825c70f35fab7, Skype pre-installed
    Documentation:            Types the given `text` into text field identified by `locator`.
    Start / End / Elapsed:    20150412 15:35:10.558 / 20150412 15:35:10.903 / 00:00:00.345
    KEYWORD: Selenium2Library.Capture Page Screenshot
    15:35:10.558   INFO   Typing text 'Skype pre-installed' into text field 'IO:f3776ac9c0a8010a003825c70f35fab7'
    15:35:10.903   FAIL   ElementNotVisibleException: Message: Element is not currently visible and so may not be interacted with
```

Figure 8. Service Catalog test suite log with an error. Screenshot [16].

The log clearly displays all of the steps in the test case and that way, it is easy to pinpoint the root cause of the issue. In the example seen in figure 8, the failed keyword is highlighted in red and an error message is displayed. Also the “id” parameter of the web element is displayed. The last keyword is a screenshot of the error situation, taken automatically by the Robot framework using Selenium’s ability to take screenshots, thus making it more straightforward to actually go and find the misbehaving element.

This example does not necessarily describe an error situation. It could also be a required feature, but the use case or the test case have not been updated accordingly. This kind of behavior in test automation is common and relates to the issues mentioned in chapter 2.2. A test automation can only find issues from places where they are told to look for them.

7.3 Conclusions

The executed tests show that it is possible to successfully automate the testing of Service Catalog. Each test case needs to be defined per Catalog Item. Of course, if one Catalog Item needs more than one use case, then each test case needs to be defined per use case. Because Service Catalog is a part of the Service Now’s GUI, it might be difficult to maintain the test automation suite, because the GUI might be updated and change even more in the next system version. Although, implementing a complete test automation suite for Service Catalog is time consuming, it might still be feasible, especially for bigger projects. After implementation, the test automation could be used for regression tests.

The overall results are mainly positive and using the Robot framework made implementing test automation easier than, for example, programming Java or Python libraries which use Selenium directly.

8 Further Development

During the implementation phase, a few development ideas emerged, which would take the Robot framework test automation even further. Unfortunately, the schedule for the project was too hectic and the scope was too small to really accommodate a larger scale test automation implementation.

One of the ideas was to name the test cases according to the defined use cases for example using an ID number on the names. This would help the maintenance and documentation, because it is possible to refer to different parts of the test automation project with one name.

Also the configured common keywords could be expanded even more. For instance, new keywords could be defined to match all different Variable types. This would speed up the test case implementation and make the test cases more human readable.

The biggest and most important development idea was to take the Selenium server into action. The use of the Selenium server would make it possible to execute tests on multiple machines or VMs simultaneously. This would make it possible to automate tests with specific browsers, browser versions or with hosts running different setups all together.

9 Discussion

This thesis has introduced the main points in test automation and how to implement a test automation suite using the Robot framework. Nowadays Service Now is a popular IT-service management platform in the corporate world and this thesis could provide a basis for many companies that are both using Service Now and considering taking test automation into action. As mentioned in chapter 8, the use of Selenium server in addition to this set of tools could provide a comprehensive Service Now testing package.

In the final year project, the test automation and its configurations worked as planned. The testing was efficient, fast and accurate. Because all of the tools used in this implementation were open source, they were also free to use. The costs in projects like this would only consist of the used work hours for defining use cases and documentation, installing the needed parts for a testing environment and implementing the test suite.

The Robot framework test automation could be used for easy regression testing and for instance to carry out mass testing. Mass testing could be run during nights and left unattended. The results could be examined and analyzed the next morning. Although the use of the Robot framework would not entirely replace human testers, it would reduce the workload considerably. If an organization is not mature enough to define precise use cases in an understandable form, it would be recommended to opt out from test automation and instead utilize human testers.

The development of the common resource file could be continued and kept on going, and it could be expanded for other applications of Service Now as well. In some point the collection of generic keywords would be huge and testing almost anything in Service Now would be easy.

10 Conclusion

As a result of the final year project, a fully working, expandable web application GUI test automation, using only free of charge tools, was successfully created. Considering that no bigger technical problems were identified during the implementation and the only obstacle was time consumption when scouring through hundreds of lines of web page source code and hundreds of pages of Selenium documentation, the project as a whole was a success.

The goals of this project were reached. Although the minimum requirements of this project were met, the research on ATDD could have been emphasized. Also more comparisons between a human and a computer tester could have been made.

All in all, test automation is an important part of bigger software development projects and it could help tackle the tedious testing phase, if built properly. Taking a step towards test automation is a huge leap for any organization and the advantages and the disadvantages should be weighed carefully.

References

1 Huizinga D., Kolawa A. Automated Defect Prevention: Best Practices in Software Management. New Jersey: Wiley-IEEE Computer Society Press; 2007.

2 Laukkanen P. Data-Driven and Keyword-Driven Test Automation Frameworks. Master’s Thesis. Helsinki University of Technology - Aalto University; 2006.

3 Introduction [online]. Robot Framework.
URL: http://robotframework.org/#introduction. Accessed 5 February 2015.

4 Nokia Solutions and Networks. Robot Framework User Guide [online]. Robot Framework; 16 January 2015.
URL: http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html. Accessed 10 March 2015.

5 Pugh K. Lean-Agile Acceptance Test-Driven Development: Better Software through Collaboration. New Jersey: Addison-Wesley; 2011.

6 Examples [online]. Robot Framework.
URL: http://robotframework.org/#examples. Accessed 14 March 2015.

7 Selenium2Library User Guide [online]. Robot Framework; 2 November 2014.
URL: https://rtomac.github.io/robotframework-selenium2library/doc/Selenium2Library.html. Accessed 15 March 2015.

8 Selenium WebDriver Documentation [online]. Selenium HQ; 4 May 2015.
URL: http://docs.seleniumhq.org/docs/03_webdriver.jsp. Accessed 15 March 2015.

9 Introduction to Service Catalog [online]. Service Now; 18 March 2013.
URL: http://wiki.servicenow.com/index.php?title=Introduction_to_Service_Catalog. Accessed 7 February 2015.

10 Service Catalog Variables [online]. Service Now; 19 January 2015.
URL: http://wiki.servicenow.com/index.php?title=Service_Catalog_Variables. Accessed 7 February 2015.

11 Client Scripts [online]. Service Now; 9 February 2015.
URL: http://wiki.servicenow.com/index.php?title=Client_Scripts. Accessed 8 February 2015.

12 Creating a Catalog Client Script [online]. Service Now; 6 May 2014.
URL: http://wiki.servicenow.com/index.php?title=Creating_a_Catalog_Client_Script. Accessed 8 February 2015.

13 GlideForm (g_form) [online]. Service Now; 30 April 2015.
URL: http://wiki.servicenow.com/index.php?title=GlideForm_(g_form). Accessed 9 February 2015.

14 Navigation and the User Interface [online]. Service Now; 12 March 2015.
URL: http://wiki.servicenow.com/index.php?title=Navigation_and_the_User_Interface#Content_Frame. Accessed 20 March 2015.

15 Service Now [computer program]. Version glide-fuji-12-23-2014__patch1-02-11-2015. Santa Clara, United States of America: Service Now; February 10, 2015.

16 Robot Framework [computer program]. Version 2.8.7. Espoo, Finland: Nokia Networks; March 30, 2015.

base-suite-test.txt

```
#
# Author: Jani Luostarinen
<jani.luostarinen@symfoni.com>
# Version: 1.0.20150212
#

*** Settings ***

Library  Selenium2Library
Resource  base-variables-test.txt
Resource  base-resource-test.txt
Resource  base-tests.txt

*** Test Cases ***
Test Login
  Test Login
```

base-variables-test.txt

```
#
# Author: Jani Luostarinen
<jani.luostarinen@symfoni.com>
# Version: 1.0.20150212
#

*** Variables ***

${baseurl}  https://INSTANCENAMEHERE.service-now.com/
${username}  robot_frmwork
${password}  **********
```

base-resource-test.txt

```
#
# Author: Jani Luostarinen
<jani.luostarinen@symfoni.com>
# Version: 1.0.20150213
#

*** Keywords ***

Login
  [arguments]  ${baseurl}  ${username}  ${password}
  Open Browser  ${baseurl}
  Unselect Frame
  Wait Until Element Is Visible  gsft_main
  Select Frame  gsft_main
  Wait Until Element Is Visible
css=div.welcome_content
  Input Text  user_name  ${username}
  Input Text  user_password  ${password}
  Click Button  sysverb_login
  Wait Until Element Is Visible  gsft_main

Logout
  Click Button  css=button.nav_header_button
  Wait Until Element Is Visible  gsft_main
  Close Browser
```

base-tests.txt

```
#
# Author: Jani Luostarinen
<jani.luostarinen@symfoni.com>
# Version: 1.0.20150213
#

*** Keywords ***

Test Login
  Login  ${baseurl}  ${username}  ${password}
  Logout
```

suite-snc-generic.txt

```
#
# Generic ServiceNow test suite.
#
# Author: Jani Luostarinen
<jani.luostarinen@symfoni.com>
# Version: 1.0.20150411
#

*** Settings ***

Library  Selenium2Library
Resource  settings-snc-demo-instance.txt
Resource  resource-snc-common.txt
Resource  tests-sc.txt
Suite Setup  Login  ${baseurl}  ${username}
${password}
Suite Teardown  Logout

*** Test Cases ***

Order Sales Laptop
  Order Sales Laptop

Order Development Laptop
  Order Development Laptop
```

settings-snc-demo-instance.txt

```
#
# Settings for ServiceNow demo instance.
#
# baseurl: Target instance url
# username: Target instance user account
# password: User account password
# family: Service Now version (Supported: Dublin,
Eureka, Fuji)
#
# Author: Jani Luostarinen
<jani.luostarinen@symfoni.com>
# Version: 1.0.20150411
#

*** Variables ***

${baseurl}  https://INSTANCENAMEHERE.service-now.com/
${username}  robot_frmwork
${password}  **********
${family}  Fuji
```

resource-snc-common.txt

```
#
# Common generic purpose keywords for interacting with
ServiceNow.
# Should not contain any test case specific
functionality.
#
# When adding or modifying functions please follow
these rules:
#
#   - Each keyword should wait that action performed
is ready
#     so next keyword has clean environment to proceed
#
#   - Never assume which frame is selected - always
select
#     correct frame just in case before doing anything
#
#   - Always test compatibility with all families
supported
#
# Supported families (SNC versios): Fuji, Eureka,
Dublin
# Author: Jani Luostarinen
<jani.luostarinen@symfoni.com>
# Version: 1.0.20150411
#


*** Keywords ***


Select Navigator
  Unselect Frame
  Wait Until Element Is Visible  gsft_nav
  Select Frame  gsft_nav
  Wait Until Element Is Visible  css=ul.nav-wrapper
```

```
Select Main Before Login
  Unselect Frame
  Wait Until Element Is Visible  gsft_main
  Select Frame  gsft_main
  Wait Until Element Is Visible
css=div.welcome_content

Select Main
  Unselect Frame
  Wait Until Element Is Visible  gsft_main
  Select Frame  gsft_main
  Wait Until Element Is Visible  css=i.pointerhand

Login
  [arguments]  ${baseurl}  ${username}  ${password}
  Open Browser  ${baseurl}
  Select Main
  Input Text  user_name  ${username}
  Input Text  user_password  ${password}
  Click Button  sysverb_login
  Select Main

Logout
  Close Browser

Open Application
  [arguments]  ${application}
  Run Keyword If  '${family}' == 'Fuji'  Select
Navigator
  Run Keyword If  '${family}' == 'Eureka'  Select
Navigator
  Run Keyword If  '${family}' == 'Dublin'  Unselect
Frame
  Input Text  filter  ${application}

Open Module
  [arguments]  ${application}  ${module}
  Open Application  ${application}
```

```
Select Navigator
Click Link  link=${module}
Select Main
${sys_target} =  Get Value  sys_target
Set Test Variable  ${sys_target}
```

tests-sc.txt

```
#
# ServiceNow Service Catalog test cases.
#
# Author: Jani Luostarinen
<jani.luostarinen@symfoni.com>
# Version: 1.0.20150411
#

*** Keywords ***

Order Sales Laptop
  Open Module  Self-Service  Service Catalog
  Select Main
  Click Link  link=Hardware
  Select Main
  Click Link  link=Sales Laptop
  Select Main
  Select Checkbox
ni.IO:3cfe1f290a0a0a6a01ee1623f4982abd
  Input Text  IO:f3776ac9c0a8010a003825c70f35fab7
Skype pre-installed
  Click Button  oi_order_now_button

Order Development Laptop
  Open Module  Self-Service  Service Catalog
  Select Main
  Click Link  link=Hardware
  Select Main
  Click Link  link=Development Laptop
  Select Main
  Select Radio Button
IO:3cf4a9d60a0a0a6a00fc892d0f6834e8  500
  Click Button  oi_order_now_button
```