Test automation framework design for Web Applications

Kamil Smuga

kamil.smuga@ieee.org

COMP40070 Term Paper

School of Computer Science and Informatics

University College Dublin

April 23, 2013

Abstract

— ABSTRACT —

1 Introduction

By definition, software is applied to automate processes that exist in a real world. To build software that solves complex problems we need software development process. Software development process contains software testing phase. When testing phase is applied differs between projects and methodologies. Whether testing will be performed manually or automatically depends on type of testing and number of other business and project specific factors. However, regression and acceptance tests are proven field to apply test automation. Test automation design is the subject of this paper.

1.1 Domain Driven Design

The vision of Domain Driven Design is being widely popularized by Eric Evans. Eric wrote popular book on the subject: *Domain-Driven Design: Tackling Complexity in the Heart of Software*. In a nutshell, DDD refers to:

- domain knowledge,
- model,
- ubiquitous language.

Each listed section applies differently to various context. The appliance in test automation is described below.

1.1.1 Domain knowledge

In order to create useful software, one needs to know what software is all about - must understand the domain. The entire purpose of the software is to enhance a specific domain. In software testing context, domain knowledge refers specifically to system under test (SUT) knowledge and understanding. A person who would be suitable is an experienced specialist with hands on experience on the product.

1.1.2 Model

A model is essential part of the software design. All thinking process about the domain is synthesized into the model. The form of sharing model details can be: use case, diagram, drawing, picture or writing. Consider 2 people engaged in creating a model: test software designer — responsible for designing a test automation framework — and domain knowledge expert. They do not go into details but provide high level model of framework based on software behavior, as shown in the Figure 1 below.

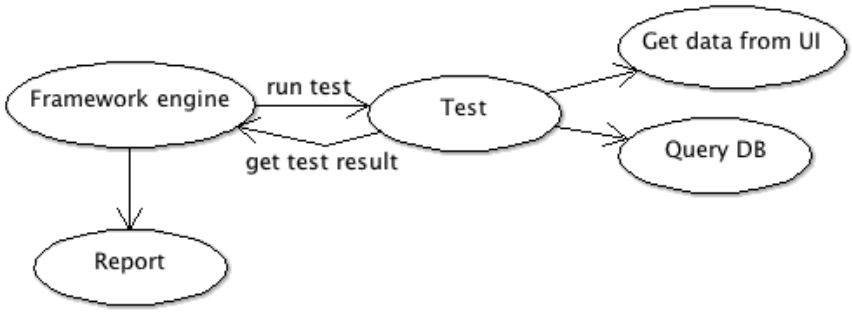

Figure 1: Test automation software model

1.1.3 Ubiquitous Language

In order to develop model of the domain, domain experts have to work with test software designers. There is a quite realistic possibility of communication barrier – software designer will think in terms of inheritance, polymorphism, OOP and domain expert will use product specific jargon. This indicates a necessity to create the same language. DDD principle suggest to use a language based on the model. This language is forced to use consistently in all forms of the communication - that is why is called the Ubiquitous Language. Furthermore, from a single test design perspective, it is crucial to express tests in the language of the end-user of application. This requirement leads to use keyword-driven framework, which details are explained below.

1.1.4 Keyword-driven framework

Keyword-driven test design abstracts the implementation of tests behind high-level actions. In UI testing, keywords represents user basic actions such as pressing keys, clicking on application tabs, typing. The tests are build as sequences of keywords. Keywords are translated into low-level action implementations. This is hidden from test writer, which makes test easier to understand and create. An example of keyword-driven framework is Robot Framework - well-known open source project widely-used in the industry. The available keywords are defined in libraries: standard (i.e. BuiltIn, OperatingSystem, String) and external (i.e. Selenium, SSH, Swing, Database). Refer to Figure 2 for test case example of jEdit file opening in Robot Framework.

Setting	Value
Library	OperatingSystem
Library	SwingLibrary

Variable	Value
${jedit}	org.gjt.sp.jedit.jEdit
${inFile}	\code\testing.cpp
${outTitle}	jEdit - testing.cpp

Test Case	Action	Argument	Argument
File opening	Start Application	${jedit}	
	Open File	${inFile}	
	Check Title	${outTitle}	

Keyword	Action	Argument	Argument
Open File	[Arguments]	${file}	
	Select Window	jEdit	
	Select From Main Menu	File|Open...	
	Select Dialog	File Browser	
	Insert Into Text Field	filename	${file}
	Push Button	Open	
Check Title	[Arguments]	${expTitle}	
	Select Window	jEdit	
	${actTitle}=	Get Selected Window Title	
	Should Be Equal	${actTitle}	${expTitle}

Figure 2: An HTML format test data file including a test case to test jEdit file opening in Robot Framework [2]

1.1.5 Model-based testing

One of the Model-based testing approaches is a generation of test cases with oracles from a behavior model. This can be seen as a specification-based test generation approach in which the model is the specification. This approach requires specific skill set from test designer — from now will be called test modeler — and good tool support. Good example of toolset is TEMA – a set of model-based testing tools designed for domain-specific GUI application testing. The toolset provides a complete set of model-based testing tools from modeling to test generation. The authors

of *Model-Based Testing with a General Purpose Keyword-Driven Test Automation Framework* paper proved that integration of TEMA and Robot Framework is feasible and effective. For high level overview refer to Figure 3.

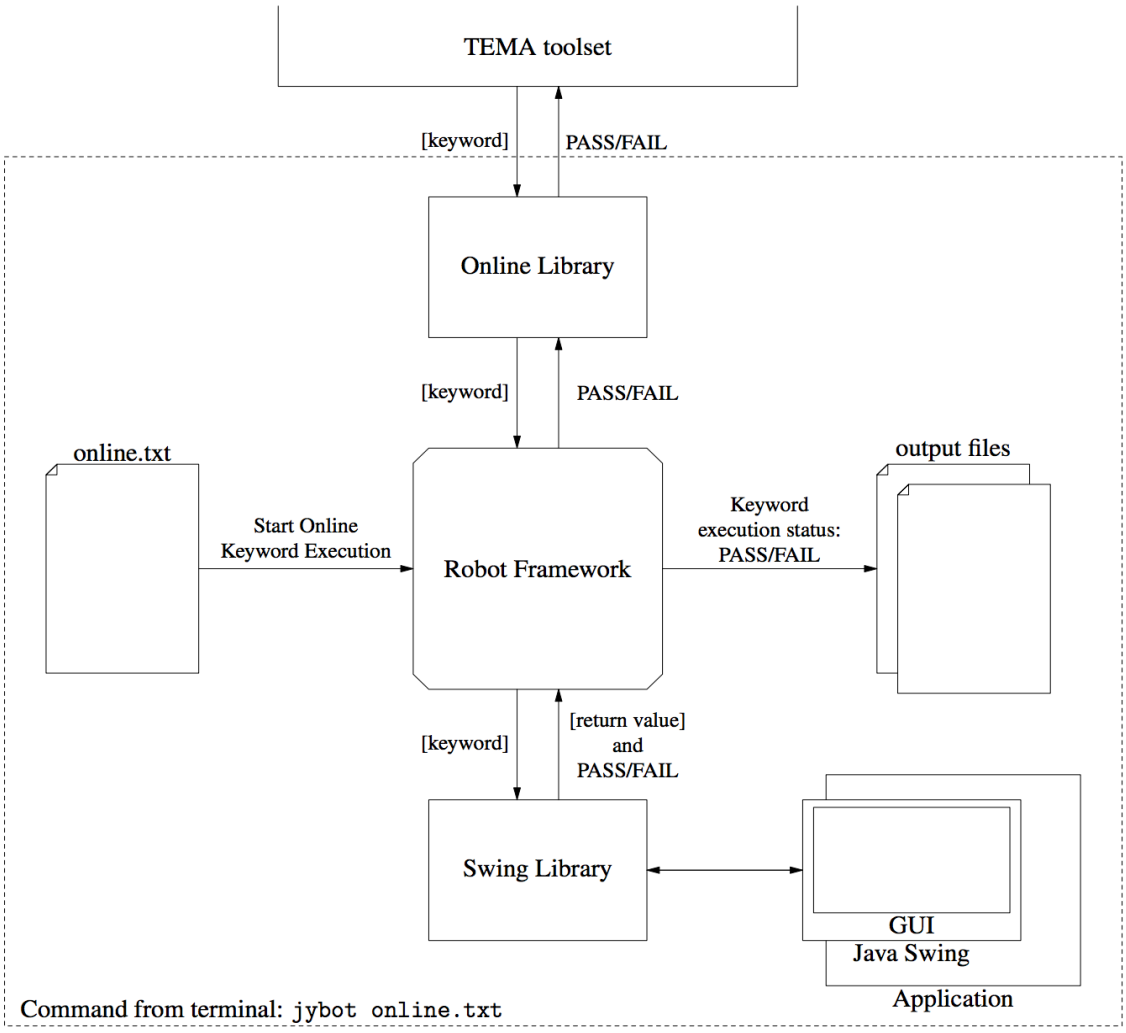

Figure 3: TEMA and Robot Framework integration [2]

Online library refers to library implemented for integration purposes – uses the internal structures of Robot framework to parse and execute the received keyword and then passes the keyword execution result to the TEMA toolset. To handle execution result exchange between *Online library* and test engine, authors of the paper implemented *Adapter unit*. For detailed information refer to [2].

Keywords almost all the keywords of the import resource files and the libraries of Robot Framework can be used in testing.

1.1.6 Make use of the "Repository" pattern to make it easier to create objects

1.2 Page Object Pattern

Page Object pattern is a design pattern used mostly in web application testing. Page Object represents application area that interacts with the test. More specifically, represents a service provided by a particular page. Code example for google.com testing:

```
class GoogleSearchPage
  def initialize
    @driver = Webdriver.new
    @q = WebElement.new
    @btn = WebElement.new
  end
  def open(url)
    @driver.get url
  end
  def close
    @driver.quit
  end
  def getTitle
    @driver.getTitle
  end
  def searchFor(searchString)
    @q.sendKeys searchString
    @btn.click
  end
  def typeSearchTerm(searchTerm)
    @q.sendKeys searchTerm
  end
  def clickOnSearch
    @btn.click
```

```
  end
end
```

The example assumes Selenium Webdriver is initialized as *driver*, *q* refers to query field and *btn* refers to search button.

This approach has a number advantages.

1.3 LoadableComponent

1.4 Bot Style Tests

1.5 Observer - for AJAX calls

1.6 Lazy Initialization

1.7 Double Checked Locking

References

[1] Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley Professional, 2003, ISBN-10: 0-321-12521-5, ISBN-13: 978-0-321-12521-7

[2] Pajunen, T.; Takala, T.; Katara, M.; *Model-Based Testing with a General Purpose Keyword-Driven Test Automation Framework*, 2011, Fourth International Conference on Software Testing, Verification and Validation Workshops

[3] Selenium wiki page, `https://code.google.com/p/selenium/w/list`