Adopting Keyword-driven Testing Framework into Jenkins Continuous Integration Tool: iProperty Group Case Study

NAZRI KAMA, YAZRIWATI YAHYA, NURULHUDA MOHAMAD AZMI, SAIFUL ADLI ISMAIL, NORZIHA MEGAT MOHD ZAINUDDIN
Advanced Informatics School
Universiti Teknologi Malaysia
Jalan Semarak, 54100 Kuala Lumpur
MALAYSIA
mdnazri@utm.my, yazriwati@utm.my, huda@utm.my, saifuladli@utm.my, norziha.kl@utm.my

Abstract: - Many organisations resort to using Agile Scrum development to shorten their time to deliver their products rapidly. In less mature Agile organisations like our case study (iProperty Group), testing is typically done by less-technical persons who possess little to no programming knowledge that contribute in specifying test cases. To our knowledge, using natural language is more descriptive and understandable to the less-technical stakeholders rather than programming oriented language. To support this situation, we propose to adopt the Robot Framework (RF) which is a native Keyword-Driven Testing (KDT) framework. The framework is integrated with existing Jenkins Continuous Integration tool in the case study. Based on our experience in the integration, the selected KDT native framework which is RF supports maintainable, reusable, understandable test assets that can be run automatically every time a new build of the product is deployed to multiple target environments. This integrated automated infrastructure allows regression testing to be executed rapidly, without sacrificing test coverage.

Key-Words: - Software Development, Software Testing, Keyword-driven Testing

1 Introduction

In Agile practices, the testing documentations are kept to minimum except for big stories or during major site revamps [1-3]. There is little effort that existing or new user stories are being tested regularly during the software development lifecycle, except for a few descriptions on the Project Tracker [3]. Factoring in rapid development lifecycle, small team and little visibility to the test processes, sometimes it could not be prevented that some of the tests would be skipped due to time constraints [4].

In less mature Agile organisations like our selected case study (iProperty Group), testing is typically done by less-technical persons who possess little to no programming knowledge that contribute in specifying test cases. To our knowledge, using natural language is more descriptive and understandable to the less-technical stakeholders rather than programming oriented language [3-5].

To support this situation, we propose to adopt the Robot Framework (RF) [6-11] which is one of a native Keyword-Driven Testing (KDT) framework. We integrate the framework with existing Jenkins Continuous Integration tool [7,8] in the case study environment. The scopes of the work are: (1) the implementation of the framework is conducted in iProperty Group Sdn Bhd Team; and (2) creation and execution of acceptance-level functional regression tests uses Selenium2Library (S2L) in Robot Framework format.

The rest of this paper is organized in following sections: Section 2 presents related keywords to the research, Section 3 explains the adoption of the RF in the case study organization, Section 4 discusses the adoption process and experiences, Section 5 evaluates the adoption process and the last concludes our contribution and future work.

2 Related Works

This section presents three main topics that are related to the study which are Web-based product development and Keyword-driven testing.

2.1 Web-based Product Development using Agile

For usage of testing tools, mainly for web-based development, JUnit is used for unit testing [6, 12-14]. In testing from user perspective for acceptance level, Selenium is documented to be widely used [6-10, 12, 13, 15,16-18,19]. Selenium is used to control
and retrieve information from web browsers, accessing web products under test. There are two variants of Selenium tools, which is Selenium IDE for capture and replay [7, 8, 20] and Selenium WebDriver [10, 18, 19] for creating the tests programmatically using popular language bindings such as C#, Java, Python and Ruby. Maintainability is a huge concern in using Selenium in Agile projects [21], particularly in Selenium IDE test scripts. Changes to web elements over the iterations may break the existing scripts. Practice such as Page Object Pattern [26] [29] is invented to be used in Selenium WebDriver test scripts. Using Page Object Pattern, definitions of elements and actions are kept in separate classes or files and will be used in the dependent test scripts. This way, if there is any change to the web elements over the course of development, only these definitions are needed to be changed and the existing tests will continue to work as usual.

Using Selenium WebDriver programmatically also allows for extensions and inclusions of other libraries. For example, SeleniumDB [18] is an extension that allows Selenium to check against information in databases during test execution. Normal Selenium tests are run manually in the browser (Selenium IDE), or from command line (Selenium WebDriver). Recent trend is running WebDriver test scripts using a test execution framework such as TestNG [28] or directly integrate them with Eclipse IDE. To further improve the visibility and traceability to the requirements, WebDriver test scripts can be further integrated with another acceptance testing framework such as FitNesse [9, 17, 22-24]. This practice is prevalent in organisations practising ATDD [22, 24].

FitNesse [9] is a wiki-based tool where acceptance tests can be defined in HTML table format. Underneath the surface, the tests are tied to corresponding library such as Selenium WebDriver. After test execution, the HTML table will be updated automatically to show which tests have passed and which have failed. This allows for visibility of test results across the stakeholders. However, the code that bridges the library and the HTML table format (called fixture) is normally written by developers [22][34], and this requires cooperation and efforts from developers as well. For mature Agile organisations, this would pose no problems as whole-team approach is being taken [25]. In contrast, for organisations those are not mature enough, testers are even expected to write unit tests, which may not be fit with their skillsets and roles as observed by Sumrell [26]. This may cause problems for new Agile teams with less-technical testers and uncooperative developers to adopt FitNesse in their environment, although there are community-made fixtures available to download.

### 2.2 Keyword-driven Testing

A related practice is Keyword-Driven Testing (KDT), which is a way of implementing and abstracting test cases in keywords [27]. Using KDT, less-technical stakeholders who possess little to no programming knowledge can still contribute in specifying test cases. An action or group of actions for test steps can be expressed in keywords, which are normally defined in natural language. Because using natural language is more descriptive and understandable to the less-technical stakeholders, it is suitable to be used in high-level tests, such as acceptance testing and GUI-based tests [28]. Hametner et al. [27] has demonstrated a case study in applying KDT on top of FitNesse [9]. Parameterisation is an essential concept for KDT to enable reuse of the same keywords using different parameter and test data [29].

Another framework, RF is a native KDT framework. These papers discuss the architecture and implementation of RF [30-32] either standalone or as part of ATDD practice. Test cases for RF can be created in text format, HTML format, and CSV (TSV) format. There is also a GUI editor for test cases called RIDE. For test execution, XML and HTML log files for test results are automatically generated by RF. As a framework created primarily for KDT, RF has a wealth of built-in keywords and libraries that interfaces with many protocols such as Operating System, String, Telnet, XML, and so on. Third party keywords and libraries such as Selenium, Android, iOS, SSH, Database and many more are also available. RF also supports tags, which can be used to select, include and exclude test cases during test execution and reporting. The usage of tags has found to be beneficial [33] in selection of test cases for execution and reporting, thus it can be leveraged in RF adoption.

### 3 Adoption of Robot Framework in the Case Study Organization

#### 3.1 Current Framework

Figure below shows the current practice and composition of iProperty.com Singapore IT team.
Referring to the above figure, for every new change (user story and/or bug fixing), the integration into the current codebase is made in stages. All the development, CI and target environments are Windows-based. There are three target environments: (1) Test: Local server with test database; (2) Beta: Restricted-access server with production database; and (3) Production: Publicly-accessible server.

Jenkins is used for source code compilation and deployment to target environments. Source codes are stored using Git in centralised repositories, which is using GitHub services.

For current process, every change in forms of source code is deployed and tested in Test environment first. Testing is done manually against three popular browsers: Internet Explorer, Mozilla Firefox and Google Chrome. After the change is verified as working as intended and does not have any adverse effects to existing functionalities, the change is then deployed to next stage, which is Beta environment, and the whole testing process is repeated until the Production environment. If there is any defect found at any stage, it depends on the discussion within the team whether the fix is to be done at which stage. For low risk and highly-urgent items, the fixes and changes can be straightaway deployed to later stages without going through the normal step by step as described earlier.

Manual exploratory testing is performed and minimal documentations are produced. This takes quite a surmountable amount of efforts for the Quality Analyst to test and retest for every environments and for every change, with little documentation specifying the correct behaviour of the functionalities.

3.2 Proposed Adoption Framework

We propose to adopt the usage of Robot Framework (RF), a Keyword-Driven Testing (KDT) framework, as shown in the following figure. Using the Selenium2Library (S2L) tool, the product are planned to be tested at the user interface level, which is nearest to the end user experience. Complex HTML, CSS and JavaScript interactions are being rendered and presented in internet browser in forms of web pages which could not be simulated at unit test level.

Using this approach, testers will create test artefacts and store them in GitHub repository. For any changes like new features or bugfixes (not shown in figure), after each deployment to testing environments (Test, Beta and Production), test execution will be triggered by Jenkins. RF will manage the test execution and uses S2L to send instructions and retrieve results from web browsers. At the end of each test execution, RF will compile test results and publishes it to Jenkins control panel, which are then will be accessible to stakeholders for further actions.

To achieve this, there are a few phases to be done. Adoption in local host for proof of concept:
- Install RF and its dependencies
- Install S2L onto RF
- Configure RF to work with Jenkins build management using plugins
• Configure Jenkins to fetch test source codes from Git repository
• Create custom keywords and test assets in RF format
• Create test cases using keywords from previous step

Using KDT may require a huge investment in time to kickstart and integrate the practice into existing infrastructure and workflow. However, the benefits of maintainable, reusable, visible, rapid and unattended test executions on multiple browsers and multiple platforms may quickly offset the initial investments.

4 Discussion
This section explains our experiences in adopting the KDT in the case study organization based on the breakdown of the planned activities.

4.1 Installation of RF and its Dependencies
RF only supports Python 2.5 and above. Python 3.x series are not supported. There are also options of using Jython on Java Virtual Machine or IronPython on .NET Framework. Since there was no restriction on the installation of Python by the current infrastructure and company policy, installation of Python was chosen to prepare for RF environment. There was also no current Python installation on the target local machine, so using Python may minimise conflicts and complexity of the final setup as opposed to using Jython or IronPython.

4.2 Installation of S2L onto RF
With the installation of pip in previous steps, installation of S2L was quite straightforward. Using one single command line, the library is automatically downloaded and installed into the default folder. However, for ChromeDriver and InternetExplorerDriver, they needed to be downloaded from their respective project websites, and manually copied into /Python/Scripts/ directory. These browser drivers are essential as interfaces to drive actions and retrieve information between S2L and the Application-Under-Test (AUT) for interoperability testing across different web browsers.

4.3 Installation of Jenkins and its Integration with RF
Primary functions of Jenkins within the company infrastructure were to compile source codes from SCM repositories and deploy the AUT to target environments. With the addition of RF, Jenkins has another role in the infrastructure as a test execution trigger.

Jenkins requires Java, thus Java SE was installed on the test machine. RF Plugin for Jenkins collects and publishes test result and artefacts from RF execution into Jenkins project folder. Historical views of test execution results can be viewed using this plugin. Detailed reporting on every test steps, and screenshots of failures at the step level can be analysed to troubleshoot both the tests as well as the AUT. Without using Jenkins plugin, execution and result can only be triggered and viewed using command line and remote desktop, which defeats the purpose of automated and unattended testing within a CI infrastructure.

4.4 Installations and Configuration of Git Client and GitHub
Because test codes are in plain text form, it is rational to treat them similar to source codes. Using Git, branching and versioning of test codes are easier to be managed. Atlassian SourceTree is chosen as a Git client because it displays branching and commits in visual form, as opposed to the native command-line-based official Git client.

Private repository for test codes is created on GitHub, a web-based Git repository hosting service. Using hosted private repository, private data and information could be restricted against unauthorised access. GitHub is a well-known Git hosting service provider, which hosted many commercial and open source projects. Historically, the uptime and security of GitHub servers were proven to be reliable. Source codes for the AUT were also stored on GitHub server, although in separate repositories.

4.5 Creation of Custom Keywords and Test Assets in RF Format
RF supports many test-writing formats: plain-text, HTML, CSV, and TSV. In extension to plain-text format, RF also supports pipe-separated format. For this project, plain-text format is used as the format to write test cases due to its simplicity and it is also
easier to be viewed in SCM tools, as opposed to using CSV or HTML formats.

Sublime Text 3 is a commercial text editor, which also has a free evaluation version. Using RF Assistant plugin for Sublime Text 3, the syntax of the RF test files can be easily viewed. Test suite settings, variables, keywords names and test steps are distinctly coloured to improve readability during test writing and test code debugging.

For reusability of test codes, custom variables are written in Python to return base URLs of the AUT in different environments and products. Using this method, a single test case can be reused many times by only passing the target environment as the parameter during the execution of the test scrip.

4.6 Creation of Test Cases Using Keywords
Due to many strategies and approaches to writing of test cases in RF, each of the approaches was used in this initial exploration. Using one single file for the test steps, custom keywords and variables may work best for small, specific and independent feature that may not be reused in other parts of other tests.

Using data-driven format was found to be very suitable for tests that share the same step, especially for features that have many business rules and expected results based on different combinations of test data.

Using separate files for test cases, variables and keywords allows for reusability for the test elements across test cases, features, and environments. However the management of coupling and dependencies for the test elements must be carefully planned to prevent high maintenance efforts.

5 Evaluation

5.1 Setup of KDT Framework for Test Status Visibility
The KDT framework had been successfully set up and integrated into the organisation’s infrastructure. The integration of RF into Jenkins provides a platform via a web interface for the stakeholders to view the currently-running test execution in addition to detailed historical view of previous test executions. Previously, test execution status could only be known by manually enquiring the Quality Analyst personnel performing manual tests.

5.2 Development of Reusable Test Assets for Better Maintainability
The test assets in forms of custom keywords, resource files and variable files were created and refactored over time. A variable file has been created serves as a global variable, reusable across all test cases in nine environments. For building blocks of each test case, custom user keywords are structured in a modular way. Each low, medium and high level keyword can be reused across test cases with minimal duplication. All of these approach results in reusable and maintainable test assets.

5.3 Creation of Self-Documenting Test Cases
Self-documenting test cases are defined as test cases that could convey their meaning and context through their names and test steps. In contrast, non self-documenting test cases are hard to be understood for both their purpose as well as their exact actions to the Application-Under-Test (AUT). Each of the test cases written in keyword form and executed using RF can be drilled down the keyword hierarchy to expose the abstraction until the lowest level if needed. At a glance, the flow of the test cases could be understood not only at the test case name level, but also of their logical decomposition when each nested layers are exposed using the HTML result file.

6 Conclusion
The adoption of KDT testing framework using RF has proven to bring net positive effects to the case study (iProperty Group) organisation. Based on the implementation, there are three main contributions of work to the organization which are: (1) Setup of KDT Framework for Test Status Visibility; (2) Development of Reusable Test Assets for Better Maintainability; (3) Creation of Self-Documenting Test Cases.

For Future works, we suggest that time measurement of test activities needs to be considered. The measurement of time taken for each test activity may provide more insights towards the efficiency of learning, adopting, integrating and
using a new framework which can be compared against metrics gathered from previous practices.

References:
[12] Dos Santos, A. M., Karlsson, B. F., Ca


