Python Test Framework with Jenkins Implementation in Software Automation Testing

Fakaruddin Fahmi Romli, Mohd Ariffudin Mohd Isa, Mohd Solehudin Abdullah, Rosnisa Abdull Razak, Sazianti Mohd Saad
Product Quality and Reliability Engineering, Mimos Berhad,
Technology Park Malaysia, 57000, Kuala Lumpur, Malaysia.
fahmi.romli@mimos.my, ariffudin.isa@mimos.my, solehudin.abdullah@mimos.my, rosnisa.razak@mimos.my, sazianti.saad@mimos.my

ABSTRACT

The Python Test Framework is a test automation tool to manage and distribute the execution of web-based functional testing. This paper presents the introduction of Python Test Framework, Jenkins and implementation of the test automation tools integrated and managed by Jenkins, the continuous integration platform, to further improve the testing efficiency as well as minimizing the effort and risk in software testing. The study were done by comparing the solution presented against the method usage of the test automation tools without the continuous integration platform and manual testing in order to gathered a concluded fact data of the test readiness effort, test execution effort, test efficiencies and the effectiveness coverage in the area of testing. The analysis showed the presented implementation able to reduce the overall testing effort up to 21% and improve the testing execution to be more manageable which eventually lower the risk for a retest probability and reduce the maintenance intervention.

KEYWORDS

Continuous Integration, Test Automation, Jenkins, Python Test Framework, Test Script.

1. INTRODUCTION TEST AUTOMATION

Test automation process involve the use of tools to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions [1]. It is common to automate a manual test process that is already in place, which is executed according to formalized and structured testing procedures. Indeed, automation is the key for the maintainability and efficacy of GUI testing, depending on the kind of requirements the test is addressing [8].
The main focus of this paper is on the use of Jenkins and Python Test Framework to run the test scripts. In a software project, there are several areas that will involve test automation such as deployment and continuous testing automation.

1.1 Deployment Automation

Automatic deployment of the build code is the main important part in Continuous Integration (CI) implementation. The codes are developed on a daily basis. Daily deployment is required to ensure that the compilation of the codes is clean and the software under development is available for pre-release. Deployment routine can also be considered as a part of the testing because it involves verification of the build codes and the readiness of the environment in accordance to the requirement for release.

1.2 Continuous Testing Automation

Continuous Testing Automation is referring to the capability of the test scripts to be executed automatically whenever required. This ability will facilitate the process to ensure and define the quality of the software under development each time new or fixed code is committed and compiled. The continuous integration platform offers the flexibilities to the tester to configure the test execution strategies that are aligned with the implemented software process.

Martin Fowler defines continuous integration as: *"a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible."* [6].

2. PYTHON TEST FRAMEWORK

The Python Test Framework, developed by Mimos Berhad, is an open source functional automation testing tool that is designed for web-based testing. It is developed in Python programming language, using Selenium webdriver API and it operates on top of Python-Nose module that acts as automation framework. The framework is capable to run multiple python test scripts across multiple browsers. The purposes of this framework are to execute functional testing automatically and to produce firm report from the outcome testing. The framework consists of four files available in bundle package (zip).

2.1 File "config.ini"

Config.ini is the main configuration file of the framework. In this file, the user will be able to set-up configuration for the browser platform, local host port, hub server port and url for the software under testing. The Selenium 2.0 webdriver remote will use these parameters when it is initialized.

2.2 File "functionR.py"

The function.py is an important configuration file that defines the function to execute the test script. This file consists of program algorithm to call the set up configuration from config.ini file and add-on customized functions from the users. A pre-defined unit test setup is available by default and is used to launch the Selenium webdriver. The function affects all the test scripts to be executed.

2.3 File "Runx.py"

This file is responsible to initiates the nosetests module to run all test cases in the current pointed test script folder and to generate the corresponding test reports. The generated test report will be in Unit test xml format and can be easily published on Jenkins without any adjustment.

2.4 File "test_google.py"

This file is a test script template for users to familiarize themselves with the standard of the python test scripts used within the Python Test Framework. The designated filename for all test cases should include the keyword "test" as the nosetests will pick test script files with a "test" keyword and execute it.

3. INTRODUCTION TO JENKINS

Jenkins is an open source continuous integration platform to manage task execution and to monitor execution of repeated jobs, like building a software project or jobs run by cron. For the purpose of the testing implementation presented in this paper, the discussion on the featured capabilities of Jenkins is narrowed down and specifically focused toward its use as the test automation support. In software test automation, among the responsibilities of Jenkins are to deploy the build code to designated server, manage test script

execution and also continually publish the testing report.
A continuous integration framework typically can provide automated source repository change detection. When changes made to the repository are detected (e.g. when developer checks in new code) a potential chain of events is put into motion [5].

3.1 Building Job

Jenkins can build and/or testing software project continuously, just like CruiseControl or DamageControl. In short, Jenkins provides an easy-to-use continuous integration system that eases the integration of changes to the project for developers and facilitates user to obtain a fresh source code build. Such an automated, continuous build capability helps to increase the productivity [2].

3.2 Monitoring

Jenkins monitors the execution of externally-run jobs like cronjobs and those that are run by a remote machine. For example, with cron, all what the user receives is regular e-mails that capture the output, and it is up to them to look at the emails diligently and notice when it broke. Jenkins keeps the outputs and aids the users to notice when something is wrong [2].

4. Jenkins Setup for Automation Testing Implimentation

Jenkins provides several types of project templates to the users to create the job. This offers a flexible project configuration based on the preferences and objectives of the user. For testing purposes, a free-style project template is selected. To further set up the project, the user has to configure the necessary configuration that includes the project information, data repositories, build triggers, build command and reporting method.

4.1 Project Information Setup

There are two essential types of information to be defined here. The first is the project title in the “Project Name” entry field. Project name data entry is not just the project title but also includes the name of this project workspace in the Jenkins local directories. By changing the name of the project, it will prompt the Jenkins to create a new workspace based on the new project name and this will result in the loss of present project data that is currently stored in the previous workspace.
The second important information to be filled is project description. Jenkins also allows users to fill the description with html expression whereby user can create a page link to the external web page and even include a picture within the project according to their own creativity. A simple example for a link to other webpage is as below:

```
<a
href=http://10.11.20.112:900/das
hboard/index/1028
target=”blank”></a>
```

4.2 Data Repositories Setup

The dedicated storage directory for the users to store related files, codes and scripts for the project usage is known as data repository. In addition, another use of data repository is to manage the versioning of the codes and scripts through branching control. Subversion SVN is the repository tools used in this study. Jenkins allows the users to configure more than one repository within a project. This feature offers compilation of integration code from different branching or version of the subversion. The link between Jenkins and repository is simply defined by the repository url to Jenkins. The access to the repository is controlled using authentication information required by Jenkins Below is the example SVN repository url:

```
Svn://10.11.10.195/project_x/tru
nk/test/functional/system_test
```

To configure the svn url in Jenkins, select the available subversion option under the “Source Code Management” category and complete the “Repository URL” entry field. The “Local module directory” entry is optional for user to define, which is used to create a subfolder in the Jenkins local workspace for project artifact storage.

4.3 Build Triggers Setup

Build triggers is an option for users to configure the build execution time to be executed automatically. In best practices for a continuous automation testing, the automation testing is often configured to be executed after the compiled code has been successfully deployed. To perform this strategy, the testing job must be configured with a “build after other projects are built” option. Under this option, an entry field for the deployment project name needs to be filled up. The objective from this setup is to let the deployment begin once the build is clean and the completion of the deployment will trigger the testing job.

4.4 Build Command Setup

Build command is the most important and crucial setup in Jenkins to execute the build task. Users can provide several types of script supported by Jenkins to execute the build task such Batch script, Ant Script, shell script and many more. The usage of the script depends on the server environment and the nature of software under development. For automation testing, there are two compulsory execution parts that users need to provide in the build script the first is the local workspace location to grab the test script and the second is the instruction to execute the test script.
In this project, windows based operating system is used as the test server and the test script are written in Python language and supported by the selenium webdriver API, which are organized and managed by the Python Test Framework.
One of the major issues faced in automation testing is data dependencies factor. To avoid such disaster, the database is restored to the default state each time before the automation testing is executed. As mentioned before, the users can configure Jenkins to run the created scripts and in this case, it is to instruct Jenkins to execute scripts to dump the database data. Jenkins will execute the scripts starting from top to bottom. Hence it is important for users to organize the scripts properly according to their priority rank.

4.5 Batch Script to Dump Default SQL Database Data

Since the test server environment is windows based, the script can be written using the batch command. Under the “Build” category, select the “Execute Windows batch command” option and fill in the batch command to dump the SQL database. An example of SQL dump command is as shown below:

```
cd ‟C:\Program Files\MySQL\MySQL
Server 5.5\bin”
mysql –u project_x –h 10.1.6.82
–pproject_x     project_x     <
‟C:\jenkins\workspace\project_x\
database\project_x.sql”
```

4.6 Batch Script to Locate the Location of the Test Script and Execute the Testing

The script still remains as a batch command and can be further divided into two parts. The first part is to define the location of the python test script in local workspace while the second part is to initiate the testing. On the previous setting up of the repository URL, it should be noted that the system will create the folder in Jenkins workspace with a name similar to the repository folder name. By referring back to step 4.2, the system will create a folder called “system_test” in the local workspace. If it is assumed that the test script is stored in a child folder named as “script” of the “system_test” folder, then the batch command expression should be as below:

```
cd script
```

In order to start the automation testing, the users need to execute the python file “Runx.py”. This file is responsible to initiate the test scripts. The batch command expression for this purpose is as follows:

```
del *.xml
python Runx.py
```

On each completion of the testing, an .xml unit test format of the report will be generated. One of the weaknesses of this generated report is that the framework is unable to overwrite it. Because of this, the report should be deleted automatically before the next cycle of the testing is initiated. That is the main reason for defining, the instruction “del *.xml” before the initiation of the “Runx.py”.

4.7 Optional Setup

To make the testing process more interactive and informatics, optional support features are available in the Jenkins for application. There are a numerous interesting support features. One of them, which are also important features, is notification of testing status. The notification can be divided into two types test report and status notification through email.
The unit test .xml report can visually display the status of each test script and the number of time of each executed test script passes or fails with the detailed technical information.
To setup the .xml unit test report, simply select the “Publish JUnit test result report” option in the “Post build Actions” category and fill in the report directory name and report file extension as below:

```
Python\*.xml
```

The email notification information content can be customized according to the preferences of users. To setup the email notification, select the “Editable Email Notification” option in the “Post build Actions” category. There are four required parts that need to be filled up by the users. They are the “Global Recipient List”, the “Content type”, the “Default Subject” and the “Default Content”. The “Global Recipient List” refers to the recipient email addresses while the type of content is selected in the “Content type”. On the other hand, the title of the email notification is defined in the “Default Subject” entry field. The following is a sample on how to generate information of the build status and build URL:

```
<p>Test Status: ${BUILD_STATUS}
<p/>
<p>Click here to view the <a
href=”${BUILD_URL}”
target=”blank”>Build
details</a></p>
```

In the “Default Content”, users can use the provided expression to automatically generate details information from the build results. The “Content Token Reference” is a feature for the users to configure the rule of email transaction initiation. The criteria of this rule are based on the status of the build. Upon completion of all configuration, click the “save” button to save.
Fig. 1 depicts the overall overview of the integration of the Python Test Framework and Jenkins in the testing process. The Python Test Framework is designed and only works for the web-based testing purpose. This means that the software under testing with this framework is a web-based type of software.

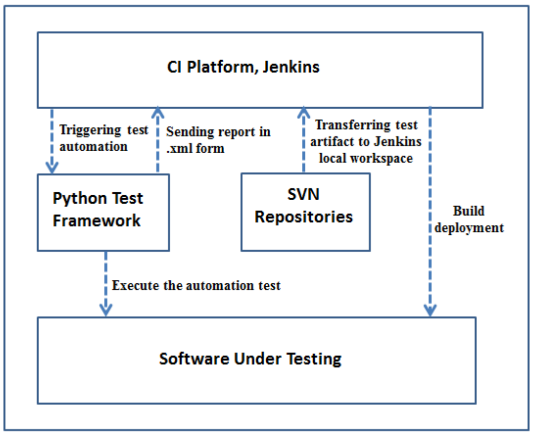

CI Platform, Jenkins
Triggering test automation
Sending report in .xml form
Transferring test artifact to Jenkins local workspace
Python Test Framework
SVN Repositories
Build deployment
Execute the automation test
Software Under Testing

Figure 1: Overview of the Higher Level Implemented Architecture

In present implementation, the latest test report is only stored in the Jenkins local workspace and not in the repository. This is due to the issues of updating and overwriting the previous .xml report. In future improvement, the capabilities will be improved to generate and bundle the history set of the test reports, and store it in the repository.

5 Effort Analysis

This section presents the analysis of Jenkins usage as a CI platform during the testing cycle. The scope of the analysis are narrowed and specified towards time consumption, testing coverage and factor of resources consumption between manual testing against automated testing with and without the CI platform. The actual values might vary but the ratio of values against each of the categories is a good representation of the actual situation in live testing implementation.

5.1 Testing Time Consumption

Analysis for time consumption begins with an assumption that only one test case and one test script is involved. Two rounds of testing and efforts of time are counted right from the beginning of test readiness until the completion of the testing.

The value is measured in hours and the comparison takes place between automation with the CI platform against the automation without the CI and manual testing.

Table 1: Time Effort Testing

Task	**Time Consumption (in hours)**		
	Automation with CI	Automation without CI	Manual Testing
Test case readiness	x	x	0.5
Test script readiness	1	1	x
Test Environment setup	2	2	2
CI configuration readiness	0.3	x	x
Test Execution	0.1	0.1	0.3
Producing Test Report	0	0	0.1
Analysis of the defect validity	0.1	0.2	0.3
Test Environment Readiness for second round of testing	x	0.2	x
Second round of testing	0.1	0.1	0.3
Producing Test Report	0	0	0.1
Analysis of the defect validity	0.1	0.2	0.3
Total	**3.7**	**3.8**	**3.9**

The data in Table 1 shows a comparison time effort in time measurement of hours for three methods of software testing implementation.

5.2 Test Case Readiness

The test case readiness is the stage where the software tester manually drafts and finalizes the test cases in text form. Test case criteria and objective are tagged with the user stories in the requirement. For automation approach, the tester abandons the traditional test cases and straight forward focuses on building the test scripts, considering the test script as a test case.

5.3 Test Script Readiness

Test script readiness phase is where the tester drafts, reviews and finalizes the test script. The effort to produce a good test script is time consuming and requires a certain level of technical knowledge to be firm in designing the test script. Nonetheless, a good test script usually has a long life span to be re-used and this reduces the risk of doing similar test script over and over again.

5.4 Test Environment Setup

In a good software testing implementation, a dedicated environment for testing is necessary to ensure a smooth testing progress without any blocking risk from an internal or external factor. Test environment set up involves the readiness of the testing servers and the entire necessary configuration as well as the services required to support the deployment of the software under testing and the testing tools. The effort for this set up process is one time only, assuming all are configured properly the first time around.

5.5 CI Configuration Readiness

CI configuration readiness is a configuration setup to enable the CI to manage and control the whole process of automation implementation. This configuration phase is the most crucial where all the integration, test environment management, deployment job and testing joy necessities to be configured precisely and firm.

5.6 Test Execution

On each implementation, the testing execution time varies from one method to the other due to their nature of testing. The automation implementation clearly takes less time due to the automated execution for every testing. In manual testing, the execution time depends on the knowledge and the experience of the tester toward the software under testing. This creates high possibility for long required testing time consumption. However, chances to discover a hidden or overlooked defect are also higher in manual testing in comparison to the automated implementation.

5.7 Producing Test Report

The automation coverage also includes auto generation of test report. It will require zero effort from the software tester and the time consumption for the tool to generate the report is within seconds. In manual testing approach, the production of test report could be a burden to the tester based on the required effort at the end of every cycle of testing.

5.8 Analysis of the Defect Validity

In the end on each cycle of testing, assuming a numbers of defects have been captured, the defects will be reviewed and analyzed and the validity of the defects will be determined. With a CI implementation, the defect validity analysis becomes much easier. In addition, the generated test log can be used by developers as references to fix the valid defect. This leads to increased efficiency of the whole development process.

6. Effort Analysis with Incremental Test Cases and Test Scripts

The effort analysis is now extended to gain more data reflecting the actual capacity of the testing. From previous time effort analysis, we can conclude the ratio of time effort for the automation with CI, automation without CI and manual testing is 0.74:0.76:0.78.

The margin of the ratio is close and small for a testing scenario involving a single test case/test script with two cycles of test. However, as the number of the test cases/test script increases, the gap becomes wider and this situation highlights the advantage for having the automation with the CI platform implementation.

The analysis consists of three parts, which summarize the hour consumption of the non-occurrence and occurrence effort in end-to-end testing process. The main data collection is based on the incremental numbers of the test cases / test scripts from 20 to 100, as displayed in Table 2. The scenario in this analysis is obtained by assuming each test case/test script is able to capture one defect.

Table 2: Time Effort in Testing Table with Increasing number of the test cases.

	Task	**Time Consumption (in hours)**		
		Automation with CI	Automation without CI	Manual Testing
PART A	Test case readiness	x	x	0.5
	Test script readiness	1	1	x
	Test Environment configuration	2	2	2
	CI configuration readiness	0.3	x	x
	Test Execution	0.1	0.1	0.3
	Producing Test Report	0	0	0.1
	Analysis of the defect validity	0.1	0.2	0.3
	Test Environment Readiness for round 2 testing	x	0.2	x
	Second round of testing	0.1	0.1	0.3
	Producing Test Report	0	0	0.1
	Analysis of the defect validity	0.1	0.2	0.3
	Total Effort	**3.7**	**3.8**	**3.9**
	Time **(Value A)**			
PART B	Total effort minus the one time setup task **(Value B)**	**1.5**	**1.7**	**1.9**
PART C	20 Test cases / test scripts **(Value B** x 19) + **Value A**	32.2	36.1	40
	40 Test cases / test scripts **(Value B** x 39) + **Value A**	62.2	70.1	78
	60 Test cases / test scripts **(Value B** x 59) + **Value A**	92.2	104.1	116
	80 Test cases / test scripts **(Value B** x 79) + **Value A**	122.2	138.1	154
	100 Test cases / test scripts **(Value B** x 99) + **Value A**	152.2	172.1	192

6.1 Analysis Part A

In "Part A", the accumulated time data is the default hour value for the test readiness, which involves two rounds of testing execution with one test case/test script.

6.2 Analysis Part B

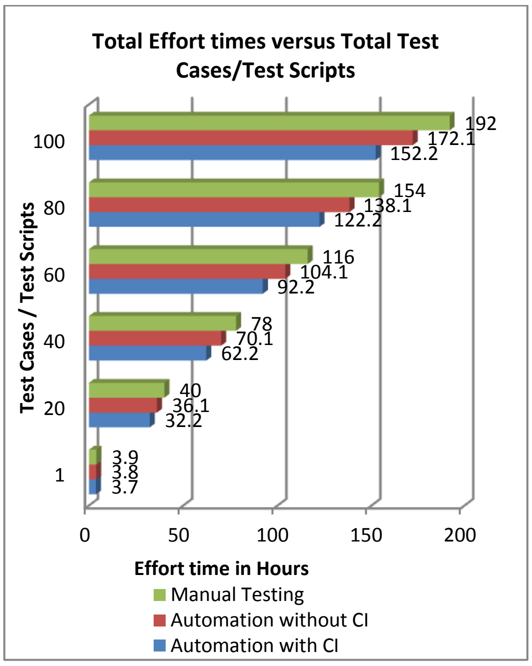
The purpose of "Part B" is to calculate the hour value for occurrence tasks, which later on will be used in "Part C" calculation. The equation used to gather that occurrence hour value is equal to the total effort values minus the non-occurrence effort values (values with the yellow highlight in the Part A).

6.3 Analysis Part C

In PART C section, the hour values is the total effort values minus the non-occurrence effort values (values with the yellow highlight in the PART A), times with the total test cases/test scripts and plus it with the total effort values (Value A). The purpose of this practice is to discover the total effort values based on the different numbers of test cases and test scripts.

6.4 Conclusion of Effort Analysis

From results in Table 2, it can be summarized that the margin for effort time consumption from three different testing implementations will increase with increasing numbers of the test cases and test scripts. To get a clearer view, Graph 1 shows a comparison between the three presented testing implementations in time effort versus the total number of the test cases and the test scripts.



Graph 1: Total Effort Times versus Total Test Cases / Test Scripts

The conclusion can be made upon the analysis data. First, the automation maintenance effort is decreasing with Jenkins' implementation as a CI platform. By referring to Table 1, Jenkins already caters and self-maintains the test environment readiness, which contributes to a saving of 0.2 hours of the effort during the second and onward cycles of test. Secondly, the Effort's weightage increases concurrently with the numbers of the test cases/test scripts. With Jenkins CI implementation, tester only need to focus on the test script whereas other important supporting tasks like environment readiness, reports, deployment and testing scheduling are handled by Jenkins.

In Graph 1, referring to the highest volume of test case/test scripts data, which is 100, the total margin of hours between automation with CI and without CI is 19.9 hours. This represents approximately 12% of time savings. The overall effort savings with the implementation test automation with CI against the manual testing is 21%.

7. Comparative Analysis

In the comparative analysis, data criteria will focus on solving the difficulties faced in software testing. The measurement values are in three types: automatic, manual and not applicable. The scoring severity is as per below:

i. Automatic : 1 point
ii. Manual : 0.5 point
iii. Not applicable : 0 point

From the comparative analysis results shown in Table 3, the score suggests that the implementation of the Python Test framework-Jenkins contributes up to **37%** benefits to the tester problem solving against the automated implementation without CI.

Table 3: Comparative Analysis

Criteria	Automation with CI	Automation without CI	Manual Testing
Test Readiness			
Test Server Management	Auto	Manual	Manual
Test Scripts/Test Cases Storing	Auto	Manual	Manual
Test Service Initiation	Auto	Manual	Not Applicable
Score	3	1.5	1

Testing Execution			
Testing trigger	Auto	Manual	Manual
Test execution	Auto	Auto	Manual
Concurrent Test management	Auto	Manual	Not Applicable
Score	3	2	1
Outcome from Testing			
Result Report production	Auto	Auto	Manual
History of Result Report	Auto	Manual	Manual
Score	2	1.5	1
Total Score	**8**	**5**	**3**

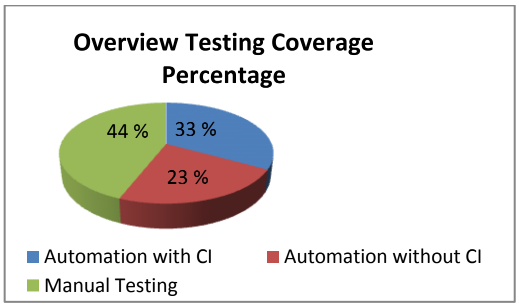
8 Testing Coverage Analysis

The testing coverage weightage are analyzed to determine the strength and the weakness of all three implementation methods presented. The analysis data is gathered through the comparison of each implementation against each other. Five types of testing coverage are used: functional, adhoc, negative, usability and deployment testing. Each category is divided equally with maximum 5 total points.

Table 4: Testing Coverage Analysis

Criteria	Automation with CI	Automation without CI	Manual Testing
Functional Testing	1	1	1
Adhoc testing	0	0	1
Negative Testing	1	1	1
Usability Testing	0	0	1
Deployment Test	1	0	0
Total Score	**3/5**	**2/5**	**4/5**

The result in Table 4 summarizes the coverage capabilities of the presented implementation on each method of testing. It can be observed that even though manual testing is laborious and time consuming, it is proven to be the most reliable in major areas of testing coverage. In other hand, the automation with CI implementation has one step advantage against the automation without the CI, which is the deployment testing. The deployment testing is one of the most important tasks in the process. It is not just to define whether the code build is clean but also to decrease the consumed time for product release readiness setup. Another advantage in having continuous deployment test is to avoid unnecessary testing efforts. With the CI implementation, the deployment timing execution can be configured flexibly by the users upon the status condition of the code compilation. Moreover, all of this is done automatically without any human intervention, giving it a great improvement and benefits against the implementation of the automation without CI and the manual testing.



Graph 2: Overview Testing Coverage Percentage

The conclusion that can be made from this analysis is that the automation is indeed saving a lot of testing efforts and times. However, manual testing is still a proven practice for full testing coverage as shown in Graph 2. The testing involves the human factors and influence such usability, ad-hoc testing and monkey testing still remains to be best done manually.

9 Contribution to PQRE

PQRE is a test department under Mimos Berhad, in charged in the product testing to assured the quality and the reliability of the released product. In the web base testing, PQRE already implemented an automation testing with a numbers of years to improve the efficiency of the testing deliverable.

With the presented implementation, the improvement of the functional testing is extended.

Table 5: Total Functional Bug Discovery during the Testing Phase

Sprint 1 Functional Test (3 days)	**Total bugs captured**	**Total fixed bugs**	**Balance unfixed bugs**
9/1/2012	**13**	13	0
10/1/2012	0	0	0
11/1/2012	0	0	0
Sprint 2 Functional Test (9 days)	**Total bugs captured**	**Total fixed bugs**	**Balance unfixed bugs**
25/1/2012	**35**	29	**4**
26/1/2012	0	0	**4**
27/1/2012	0	0	**4**
30/1/2012	0	3	**1**
31/1/2012	0	0	**1**
1/2/2012	**5**	3	**3**
2/2/2012	0	2	**1**
3/2/2012	0	1	0
6/2/2012	0	0	0
Sprint 3 Functional Test (4 days)	**Total bugs captured**	**Total fixed bugs**	**Balance unfixed bugs**
7/2/2012	**15**	15	0
8/2/2012	0	0	0
9/2/2012	0	0	0
10/2/2012	0	0	0

Table 5 indicated the total numbers of the system functional defects discovered during the testing phases of the actual project under development. The project genre is a web base platform and having a features enhancement in a certain area of the components with a medium development weightage. However the details of the project are excluded to maintain the company trade secret policy. The data in the table presented three (3) significant improvements in the functional test deliverable.

The first improvement is the automation testing are well maintained and managed thus assured the smoothness of the testing progress on daily continuous testing execution.

The second improvement is the defect is captured earlier and furthermore the produced test report are included the fail information logs which is sufficient enough to be a references allowed the developers efficiently fixing the issues earlier and faster.

The third improvement is the Sprint time saving. The Sprint duration presented in the table is dedicated for the testing and bug fixing tasks. As we can see there were 6 days of the software under testing is free from any defects and from that we can conclude that the project deliverable are 6 days ahead of schedule. In percentages value, the testing duration saved in days is 37.5%.

10 ACKNOWLEDGMENT

Many thanks to all staffs of PQRE department for encouraging the automation implementation. Also, thanks to the founder of the Python Test Framework, Fairul Rizal Fahrurazi, for delivering a wonderful automation solution tools. A special thanks to Dr. Fairuz Izzuddin Romli from Engineering Faculty of University Putra Malaysia for his comments on this technical paper, and also to Mr. Redzuan Abdullah, Director of PQRE, for reviewing this technical paper.

11 CONCLUSIONS

This study proposed an approach for increasing the effort saving rate and improving the efficiency in software testing through Python Test Framework as a test automation and Jenkins as a continuous integration implementation. The analysis data shows that the automation testing has advantages in decreasing the effort and time consumption of the software testing with about 10% of saved time against manual testing. The benefits are extended further by applying continuous integration platform to make the automation testing more organized, manageable and overall, this approach can save up to 21% of the testing effort against the manual testing.

Think of the CI server as a way of taking the grunt work out of checking in code. Machines are good at executing repeated tests, people aren't [7]. The manual testing is proven to be a laborious and requires a luxury amount of manpower to maintain and execute the testing in order to balance the spending effort in working hours. However, the overall coverage from manual testing can deliver a released software with a near zero defects. On the other hand, the automation testing implementation still has a limitation in testing coverage. Moving forward in the future of automation improvement, the research will focus on the ability to adapt with human factors in order to cover two major testing areas, which is the usability testing and the adhoc testing.

12 References

1. M.N. Alam,"Software Test Automation Myths and Facts", benchmarkqa publication archive (2007), http://www.benchmarkqa.com/pdf/papers_automation_myths.pdf
2. Jenkins Wikipedia, "What is Jenkins?", Kohsuke Kawaguchi, https://wiki.jenkins-ci.org/
3. Michael Bolton.; Rapid Software Testing. Agile Conference, Berlin , Germany(2010).
4. Python Test Framework Wikipedia, Fairul Rizal Fahrurazi, https://github.com/fairul82/PythonTestFrame work1.0/wiki
5. Stolberg.S.; Enabling Agile Testing through Continuous Integration. Agile Conference, AGILE '09 , pp. 369--374, IEEE Conference Publication, Chicago (2009).
6. "Continuous Integration", Martin Fowler, http://www.martinfowler.com/articles/continuousIntegration.html
7. Ade Miller.; A Hundred Days of Continuous Integration. Agile Conference, AGILE '08, pp. 28--293, IEEE Conference Publication, Toronto (2008).
8. Bertoli.C, Mota.A.; A Framework for GUI Testing Based on Use Case Design. Third Software Testing, Verification and Validation Workshop (ICSTW), pp. 252--259, IEEE Conference Publication, Paris (2010).