JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 631-648 (2018)
DOI: 10.6688/JISE.201805_34(3).0004

A System for Composing and Delivering Heterogeneous Web Testing Software as a Composite Web Testing Service*

SHIN-JIE LEE[1,3], YOU-CHEN LIN[2], KUN-HUI LIN[2] AND JIE-LIN YOU[3]
[1]*Computer and Network Center*
[2]*Institute of Manufacturing Information and Systems*
[3]*Department of Computer Science and Information Engineering*
National Cheng Kung University
Tainan, 701 Tainan
E-mail: jielee@mail.ncku.edu.tw; {t824675951535; ashlin1112}@hotmail.com; ygl0118@gmail.com

Load testing and cross-browser testing are ones of the web testing types that heavily rely on the support of cloud computing platforms for realizing the concept of TaaS (Testing as a Service). Although efforts have been made on composing heterogeneous web services, little emphasis has been put on composing heterogeneous web testing tool services and delivering a composite web testing service as a whole. The main challenge involved in the composition of heterogeneous web application testing tools is the incompatibility of their inputs and outputs. However, the need to manually configure the tools greatly undermines the convenience and applicability of their applications. In this paper, we propose a system for the composition and delivery of heterogeneous web testing tools with the following key features: (1) four adapters to automatically bridge the gap between the inputs and outputs of six state-of-the-art testing tools; (2) a composite web testing service with the adapters; and (3) two adapters to enable delivering the composite service via emails. Experimental results demonstrate the effectiveness of the proposed system in reducing the effort required for load testing and cross-browser testing in comparison with a conventional method.

Keywords: testing as a service, web application testing, testing tools composition, testing service delivery, load testing, cross browser testing

1. INTRODUCTION

In recent years, the rapid growth of cloud computing technologies has been driving research in the area of testing as a service (TaaS) [1-5]. Among the various types of web application testing, load testing and cross-browser testing rely particularly on the support of cloud computing platforms for the implementation of TaaS.

Load testing involves the application of ordinary stress to a software system to characterize system performance under normal conditions. Four types of software are generally used in the load testing lifecycle: (1) a test-case recording software, such as Badboy, which makes it easier to generate test cases using actions similar to those performed in web browsing; (2) a test-case execution software, such as JMeter, to conduct test cases with simulations involving a large number of concurrent users; (3) a system resource monitoring software, such as Cacti, which provides information related to the

Received January 28, 2017; revised March 16, 2017; accepted March 26, 2017.
Communicated by Ren-Hung Hwang.
* This research is sponsored by Ministry of Science and Technology under the grant 105-2221-E-006-154-MY3 in Taiwan.

system footprints used for running test cases; and (4) a time tracking software, such as Xdebug, which provides computation time logs of the programs used for performance tuning. Cross-browser testing is conducted for the evaluation of web applications with the aim of checking cross-browser compatibility using real browsers on a variety of platforms. Selenium [6] is a popular open-source cross-browser testing tool, which enables the recording of test cases using a Firefox extension as well as running test cases with actual browsers.

Considerable effort has gone into composing heterogeneous web services [7-11]; however, little emphasis has been put on the composition and delivery of web application testing software aimed at facilitating load testing and cross-browser evaluations. The characteristics of the open environments (in which heterogeneous web application testing software interacts) span system boundaries and operating systems; however, the existing paradigm of software usage requires considerable improvement. The main challenge involved in the composition of heterogeneous web application testing software is that *inputs and outputs of heterogeneous web testing software are often incompatible*. For example, a test case recorded by Badboy cannot be imported directly into JMeter without modifying the information related to the environment of the load testing software, as well as the footprints report generated by Cacti cannot be aligned automatically with the time period of the load testing performed by JMeter. The need to manually configure the tools greatly undermines the convenience and applicability of their applications.

Rather than directly targeting all existing web testing software, we focused on the development of a system for the composition and delivery of six state-of-the-art web testing software for use as a composite web testing service. By narrowing the focus, this work was able to deal more effectively with the major challenge. Issues related to the management of test cases and system security are beyond the scope of this paper. The system has the following key features:

- *Four adapters have been devised to bridge the gap between the inputs and outputs of six web testing software. These adapters enable the following conversions*. A raw test case recorded by Badboy can be automatically converted into an executable test case to be run by JMeter. A raw test case recorded by Selenium IDE can be automatically converted into an executable test case to be run by Selenium WebDriver. The execution time of JMeter can be determined automatically for use as an input of Cacti for generating system footprints reports. It can also be used as an input of Xdebug for the generation of time tracking reports.
- *A composite web testing service has been developed for the automated composition of the web testing software*. We developed a composite web testing service based on the continuous integration framework presented by Jenkins, as well as the proposed adapters to be used as an interface to invoke web testing software.
- *The composite web testing service can be delivered via email using two primary components*. To reduce the effort involved in the manipulation and configuration of testing software, the user is able to initiate the testing process simply by sending raw test cases to the system via email for processing. Raw test cases in this email are automatically converted and scheduled for execution by the system. Following execution, the user receives a compiled test report, again via email, which includes data related to load testing, cross-browser testing, system footprints and time tracking results.

An experiment was conducted using a real-world web application to evaluate the proposed system. Our experiment results demonstrate the effectiveness of the proposed system in reducing the effort required to perform load testing and cross-browser testing. The proposed system was shown to reduce the time required for load testing by 85.5%, compared to a conventional method. The time required for cross-browser testing was reduced by 96.7%.

The paper is organized as follows. Section 2 describes the related work. The proposed system is fully described in Section 3. Section 4 presents our experiment results. The benefits of the proposed system are outlined in Section 5

2. RELATED WORK

We collect here a number of promising research with a view toward highlighting the issues in web application testing and web service testing.

Tao and Gao [5] present a system known as Mobile TaaS as an infrastructure-as-a-service (IaaS) for mobile testing, including mobile web application testing. The system supports mobile TaaS by providing mobile instances such as server machines, mobile hubs, devices and emulators for testing environment and applications. Hsieh *et al.* [12] proposed a cloud testing service combining cloud computing and multi-testing tools. Using the proposed service, a user can deploy the testing clients in different countries to access the testing targets. When the testing is finished, the user can easily get the summarized test results.

Kamra *et al.* [13] proposed an automation testing tool by using the Google App Engine. This method provides an efficient and accurate solution for the calculation of average CPU time of a customer's requests. It will be useful in the services performance prediction of cloud computing. Wang *et al.* [14] proposed an eclipse-based load testing tool, called Load Testing Automation Framework (LTAF), for enabling performing load testing of web applications easily and automatically. A usage model is proposed to simulate users' behavior realistically in load testing of web applications, and another relevant workload model is proposed to help generate realistic load for load testing.

Hamed *et al.* [15] proposed a performance testing approach for web applications early in the development process by using two different tools: Parasoft WebKing and Hewlett-Packard LoadRunner. The results indicate that Java EE platform performs better than .NET platform in terms of response time and memory utilization. Gao *et al.* [16] focused on migrating conventional load testing tools to the cloud, for which the two significant issues are about multi-tenancy and load simulating resource management. They proposed a four layer model for cloud-based load testing, along with the approach of test request admission control and scheduling to solve these issues.

Candea *et al.* [17] presented three classes of testing services: TaaSD for developers to more thoroughly test their code, TaaSH for end users to check the software they install, and TaaSC certification services that enable consumers to choose among software products based on the products' measured reliability. Cervantes [18] proposed a test automation framework that can help a tester efficiently develop end-to-end automated test solutions. With end-to-end test automation, a tester can schedule tests to run autonomously. While tests are running, testers can utilize the time saved by performing any requisite manual testing, or developing additional automated test cases to increase test coverage.

Banzai *et al.* [19] proposed a software testing environment, called D-Cloud, which can automatically configure test environments, execute tests, and automatically inject faults into hardware devices in a virtual machine. Using D-Cloud, a tester can execute a program test for a distributed system in appropriate environments according to a scenario written in XML.

Gambi *et al.* [20] proposed a tool for automatic testing of cloud-based elastic systems, called AUToCLES. Given specifications of the test suite and the system under test, AUToCLES implement testing as a service (TaaS). It automatically instantiates the SUT, configures the testing scaffoldings, and automatically executes test suites. Dallmeier *et al.* [21] presented a technique to automatically generate tests for Web 2.0 applications. Their approach systematically explores and tests all distinct functions of a web application. The only requirements to use WEBMATE are the address of the application and, if necessary, user name and password.

Yan *et al.* [22, 23] proposed a WS-TaaS, a load testing platform for web services, which enables load testing process to be as close as possible to the real running scenarios. In this way, they aim at providing testers with more accurate performance testing results than existing tools. Li *et al.* [24] proposed an architecture for a lightweight and scalable test platform that can be used for evaluating automatic composition methods. Using this test platform will improve the development of semantic web service composition significantly.

Tsai *et al.* [25] proposed an approach to manage services on the cloud for facilitating service composition and testing. The paper uses service implementation selection to facilitate service composition similar to Google's Guice and Spring tools, and apply the group testing technique to identify the oracle, and use the established oracle to perform continuous testing for new services or compositions.

Felderer *et al.* [26] proposed a tool to support model-driven system testing of service-oriented systems in a test-driven manner by modeling the system and its tests in parallel based on the requirements. The approach is capable of test-driven development on the model level, and guarantees high quality system and test models by checking consistency and coverage.

Bluemke *et al.* [27] proposed a tool (named WSDLTest) for automatic testing of web services. WSDLTest is a simple application supporting unit testing and can test a variety of web services which are described in WSDL. The tool can be used for testing of web services for which WSDL 1.1 or WSDL 2.0 document are available.

There is a significant body of work focused on dealing with an exhaustive set of issues of web service testing and web application testing. However, there is still a lack of widespread of development of systems on composing heterogeneous web application testing software for reducing a tester's effort on conducting web application testing, which motivates us to develop a composite web testing service for assisting a tester in conducting load testing and cross-browser testing over a web application.

3. THE PROPOSED SYSTEM

Fig. 1 shows the system architecture. The system composes the following six web testing software: Badboy, JMeter, Cacti, and Xdebug, Selenium IDE, and Selenium Web-

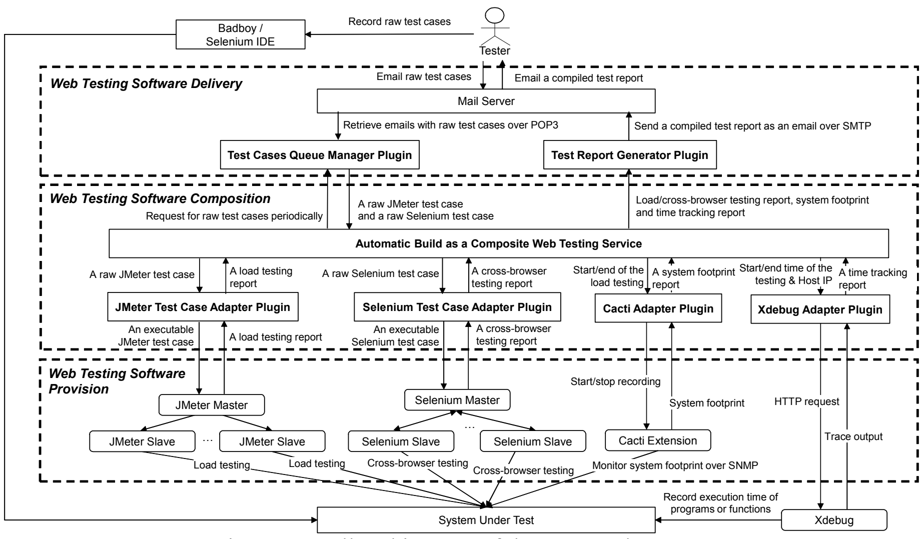

Badboy / Selenium IDE
Record raw test cases
Tester
Web Testing Software Delivery
Email raw test cases
Email a compiled test report
Mail Server
Retrieve emails with raw test cases over POP3
Send a compiled test report as an email over SMTP
Test Cases Queue Manager Plugin
Test Report Generator Plugin
Web Testing Software Composition
Request for raw test cases periodically
A raw JMeter test case and a raw Selenium test case
Load/cross-browser testing report, system footprint and time tracking report
Automatic Build as a Composite Web Testing Service
A raw JMeter test case
A load testing report
A raw Selenium test case
A cross-browser testing report
Start/end of the load testing
A system footprint report
Start/end time of the testing & Host IP
A time tracking report
JMeter Test Case Adapter Plugin
Selenium Test Case Adapter Plugin
Cacti Adapter Plugin
Xdebug Adapter Plugin
An executable JMeter test case
A load testing report
An executable Selenium test case
A cross-browser testing report
Web Testing Software Provision
Start/stop recording
System footprint
HTTP request
Trace output
JMeter Master
Selenium Master
JMeter Slave
JMeter Slave
Selenium Slave
Selenium Slave
Cacti Extension
Load testing
Load testing
Cross-browser testing
Cross-browser testing
Monitor system footprint over SNMP
Record execution time of programs or functions
System Under Test
Xdebug

Fig. 1. Overall architecture of the proposed system.

Driver. Badboy is a test case recording software for load testing. It supports exporting the recorded test cases to JMeter readable test case files. Apache JMeter is a load testing software. It provides a GUI configuration interface and supports distributed load tests. Cacti is a system resource monitoring software and provides an interface for visualizing system footprint. Xdebug, as a PHP extension, can record execution time of a PHP program or function. Selenium WebDriver is a suite of web UI testing software specifically for running test cases with most existing web browsers. Selenium IDE provides a record/playback function for authoring Selenium test cases.

A tester uses Badboy test case recorder and Selenium IDE to record raw test cases for testing the target system, and then exports them into files in JMeter and Selenium readable file formats, respectively. Subsequently, the tester sends an email with the test case files as attachments to a particular email address of the system. After the mail is sent, the test cases queue manager connects to the mail server over POP3 protocol, and then retrieves unread mails with the attached test case files from the inbox of the email address. Meanwhile, the queue manager creates a folder for each email and stores the test case files of the email into the folder. The order of the test case files of the retrieved emails for processing is determined by their sending time, and the emails' IDs are recorded in a queue. Jenkins continuous integration framework will periodically create a build with a raw JMeter test case and a raw Selenium test case obtained from test cases queue manager, and then pass the raw JMeter test case and the raw Selenium test case to JMeter test case adapter and Selenium test case adapter, respectively. In addition, Cacti adapter will be invoked to start recording the system footprint.

JMeter test case adapter converts the raw test case into an executable test case to be run in the load testing environment. The executable test case will be sent to the JMeter master, and then all of the JMeter slaves simultaneously simulate a number of users sending requests to the target system. The start and end time will be recorded for generating the system footprints report. Selenium test case adapter converts the Selenium test case to an executable test case into be run in the cross-browser testing environment. The executable test case will be sent to the Selenium master, and then all of the Selenium slaves will automatically invoke browsers to test the web UI components of the target system.

Cacti adapter enables Cacti to start monitoring the target system once the executable JMeter test case is executed, and to stop the monitoring while a load testing report is returned. In the system, Cacti adapter continuously monitors the system footprints over SNMP polling every 10 seconds. The system footprints include the information about the performance of CPU, memory, disk I/O, and network traffic. After the load testing is finished, the start and end time of the load testing together with the target system's host IP and port are passed to Xdebug adapter. The adapter will parse the time tracking logs generated by Xdebug hosted in the target system, and identify the longest time record as a time tracking report.

The test report generator generates a compiled test report as an email with URL links referring to the load testing and cross-browser testing reports, the system footprints and the time tracking report. The email will be sent to the tester over SMTP protocol. The tester can receive the email and see the compiled test report.

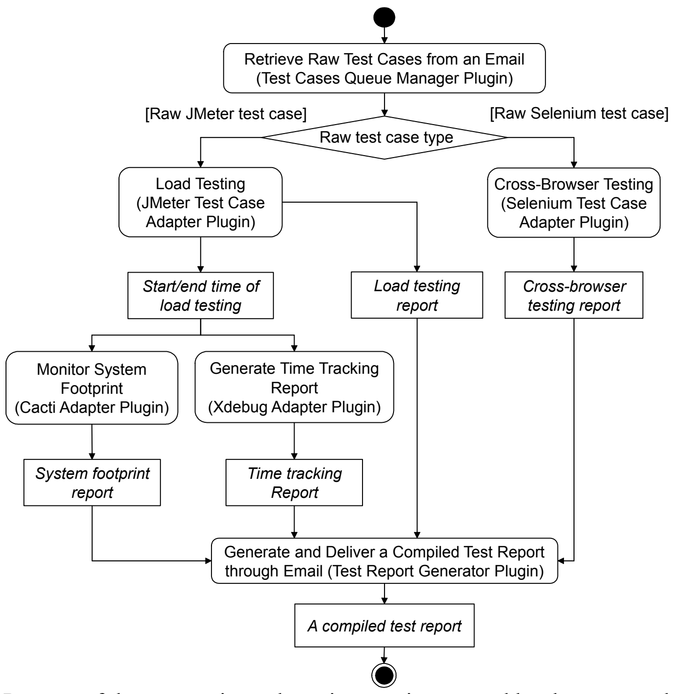

Retrieve Raw Test Cases from an Email (Test Cases Queue Manager Plugin)
[Raw JMeter test case]
[Raw Selenium test case]
Raw test case type
Load Testing (JMeter Test Case Adapter Plugin)
Cross-Browser Testing (Selenium Test Case Adapter Plugin)
Start/end time of load testing
Load testing report
Cross-browser testing report
Monitor System Footprint (Cacti Adapter Plugin)
Generate Time Tracking Report (Xdebug Adapter Plugin)
System footprint report
Time tracking Report
Generate and Deliver a Compiled Test Report through Email (Test Report Generator Plugin)
A compiled test report

Fig. 2. Process of the composite web testing service created by the proposed system.

Fig. 2 shows the process of the composite web testing service constructed by the system. The details of the inputs and outputs of the six developed Jenkins plugins are presented in Table 1. The inputs and outputs are implemented as Jenkins system variables based on Hudson APIs and can be globally shared among plugins.

Table 1. Interface specification of the six developed Jenkins plugins.

Jenkins Plugin	Input & Output
Test Cases Queue Manager Plugin	Output: • SAVE_DIRECTORY – The file path at which Test Cases Queue Manager places test case files. • MAIL_ADDRESS – The mail address of the sender
JMeter Test Case Adapter Plugin	Input: • SAVE_DIRECTORY – Variable created by Test Cases Queue Manager. Output: • HOST_NAME – IP address retrieved from the JMeter test case file. • HOST_PORT – Port number retrieved from the JMeter test case file. • START_DATE – The start time of the load testing. • END_DATE – The end time of the load testing. • LOAD_TESTING_REPORT – Load testing report.
Cacti Adapter Plugin	Input: • HOST_NAME – Variable created by JMeter Test Case Adapter. • START_DATE – Variable created by JMeter Test Case Adapter. • END_DATE – Variable created by JMeter Test Case Adapter. Output: • SYSTEM_FOOTPRINT_REPORT – System footprints report.
Xdebug Adapter Plugin	Input: • HOST_NAME – Variable created by JMeter Test Case Adapter. • HOST_PORT – Variable created by JMeter Test Case Adapter. • START_DATE – Variable created by JMeter Test Case Adapter. • END_DATE – Variable created by JMeter Test Case Adapter. Output: • TIME_TRACKING_REPORT – Time tracking report.
Selenium Test Case Adapter Plugin	Input: • SAVE_DIRECTORY – Variable created by Test Cases Queue Manager. Output: • CROSS_BROWSER_WEB_UI_TESTING_REPORT–Cross-browser testing report.
Test Report Generator Adapter Plugin	Input: • MAIL_ADDRESS – Variable created by Test Cases Queue Manager. • LOAD_TESTING_REPORT – Variable created by JMeter Test Case Adapter. • CROSS_BROWSER_WEB_UI_TESTING_REPORT – Variable created by Selenium Test Case Adapter.

3.1 Test Cases Queue Manager

Test cases queue manager periodically retrieves testers' emails from a mail server and extracts attached test case files. Fig. 3 shows the operational concept of receiving test cases by the test cases queue manager. The process involves the following key steps. Firstly, the manager configures the POP3 properties in order to connect to a mail server. In this work, a particular email account is created for receiving test cases sent from testers. Secondly, the manager retrieves the unread mails from the inbox of the account. Thirdly, the manager creates a folder for each email. At last, the manager extracts attached JMeter or Selenium raw test cases from each unread email, and each email ID will be added into a queue. While the Jenkins creates a build and requests the queue manager for test cases, the email ID at the front of the queue will be removed and the corresponding folder information will be passed to JMeter test case adapter or Selenium test case adapter.

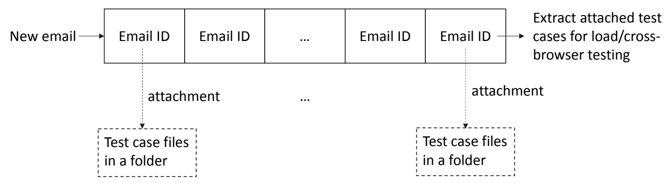

New email
Email ID
Email ID
...
Email ID
Email ID
Extract attached test cases for load/cross-browser testing
attachment
...
attachment
Test case files in a folder
Test case files in a folder

Fig. 3. Operational concept of test case queue manager.

3.2 JMeter Test Case Adapter

While deploying the JMeter load testing environment, two properties in the JMeter property file were modified. One property is "Remote Hosts". In order to simulate up to 1000 simultaneous users, we set this property to: "remote_hosts = Slave_IP1:1099, Slave_IP2:1099, Slave_IP3:1099, Slave_IP4:1099" to enable the distributed testing mode. The second property is "Results file configuration". The legitimate values are xml, csv and db, but only xml and csv are currently supported by JMeter. The original property is changed from "jmeter.save.saveservice.output_format=csv" to "jmeter.save.saveservice.output_format=xml".

As the adapter receives a raw JMeter test case, the adapter will convert the raw test case to a number of executable test cases for simulating different numbers of concurrent users. In this work, four executable test cases will be generated based on a raw test case for simulating 100, 200, 500, 1000 concurrent users. In a generated executable test case, three attributes are overridden: Thread Group number, HTTP Sampler – Follow Redirects, and HTTP Sampler – Auto Redirects.

Table 3 shows an executable test case generated from a raw test case of Table 2. The value of ThreadGroup.num_threads is changed to 100, and the values of HTTPSampler.follow_redirects and HTTPSampler.auto_redirects are changed to true and false, respectively.

Table 2. An example of a raw JMeter test case.

```
<!– Original test case file: test_case.jmx-->...
<ThreadGroup guiclass="ThreadGroupGui" ...>
    <stringProp name="ThreadGroup.num_threads">1</stringProp> ...
</ThreadGroup>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" ...>
    <boolProp
  name="HTTPSampler.follow_redirects">false</boolProp>
    <boolProp
  name="HTTPSampler.auto_redirects">true</boolProp> ...
</HTTPSamplerProxy>
```

Table 3. An example of a generated executable JMeter test case.

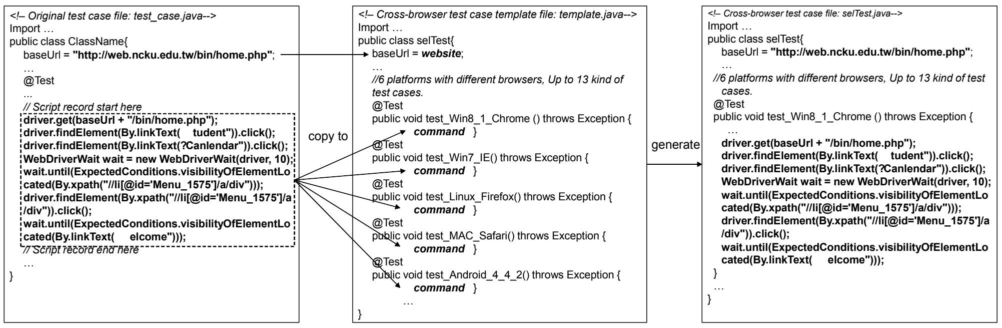
```
<!– Executable test case file: test_case_100.jmx--> ...
<ThreadGroup guiclass="ThreadGroupGui" ...>
    <stringProp
  name="ThreadGroup.num_threads">100</stringProp>...
</ThreadGroup>
<HTTPSamplerProxy guiclass="HttpTestSampleGui" ...>
    <boolProp
  name="HTTPSampler.follow_redirects">true</boolProp>
    <boolProp
  name="HTTPSampler.auto_redirects">false</boolProp>...
</HTTPSamplerProxy>
```

After the executable test case is generated, the following command will be called to conduct distributed load testing in windows console mode: "jmeter-n-t %PROJECT%-l %OUTPUT%-r". Subsequently, the JMeter master dispatches the executable test cases to the slaves and manages the load testing. After the load testing is finished, the master will receive all the test results from the slaves, and send the results as a load testing report to the adapter.

3.3 Selenium Test Case Adapter

We deployed a Selenium grid testing environment using Selenium server API. In the Selenium master server, the following command is used to establish a Selenium Grid con-sole: "java-Dfile.encoding=UTF-8-cp json.jar; selendroid-grid-plugin.jar; selenium-server.Jar org.openqa.grid. selenium.GridLauncher-capabilityMatcher io.selendroid.grid. SelendroidCapabilityMatcher-role hub". The heterogeneous Selenium slave servers need to register to the Selenium master. In this work, various operation systems, including Windows, Linux, OS X, and Android, were hosted on the slave servers. Four different commands are designed for registering different platforms as nodes under the Selenium master's control.

Fig. 4 shows an example of a raw Selenium test case, the executable test case template, and a generated executable Selenium test case. The attribute baseUrl represents the base URL of the target system, and the text "website" in the template will be replaced by the value of the attribute baseUrl in the raw test case. All of the texts of commands will be replaced by the code in between the script comments in the raw test case. After the executable Selenium test case is generated, it will be sent to the Selenium master, and then the master will start cross-browser testing with the executable test case. Once the testing is completed, the Selenium test case adapter will receive a cross-browser testing report and then set the report as an output variable.



```
<!– Original test case file: test_case.java-->
Import …
public class ClassName{
    baseUrl = "http://web.ncku.edu.tw/bin/home.php";
    …
    @Test
    ...
    // Script record start here
    driver.get(baseUrl + "/bin/home.php");
    driver.findElement(By.linkText(    tudent")).click();
    driver.findElement(By.linkText(?Canlendar")).click();
    WebDriverWait wait = new WebDriverWait(driver, 10);
    wait.until(ExpectedConditions.visibilityOfElementLo
    cated(By.xpath("//li[@id='Menu_1575']/a/div")));
    driver.findElement(By.xpath("//li[@id='Menu_1575']/a
    /div")).click();
    wait.until(ExpectedConditions.visibilityOfElementLo
    cated(By.linkText(     elcome")));
    // Script record end here
    …
}
```

```
<!– Cross-browser test case template file: template.java-->
Import …
public class selTest{
    baseUrl = website;
    …
    //6 platforms with different browsers, Up to 13 kind of
    test cases.
    @Test
    public void test_Win8_1_Chrome () throws Exception {
            command  }
    @Test
    public void test_Win7_IE() throws Exception {
            command  }
    @Test
    public void test_Linux_Firefox() throws Exception {
            command  }
    @Test
    public void test_MAC_Safari() throws Exception {
            command  }
    @Test
    public void test_Android_4_4_2() throws Exception {
            command  }
          …
}
```

```
<!– Cross-browser test case file: selTest.java-->
Import …
public class selTest{
    baseUrl = "http://web.ncku.edu.tw/bin/home.php";
    …
  //6 platforms with different browsers, Up to 13 kind of test
    cases.
  @Test
  public void test_Win8_1_Chrome () throws Exception {
      …
    driver.get(baseUrl + "/bin/home.php");
    driver.findElement(By.linkText(    tudent")).click();
    driver.findElement(By.linkText(?Canlendar")).click();
    WebDriverWait wait = new WebDriverWait(driver, 10);
    wait.until(ExpectedConditions.visibilityOfElementLo
    cated(By.xpath("//li[@id='Menu_1575']/a/div")));
    driver.findElement(By.xpath("//li[@id='Menu_1575']/a
    /div")).click();
    wait.until(ExpectedConditions.visibilityOfElementLo
    cated(By.linkText(     elcome")));
  }
  …
}
```

copy to
generate
A raw Selenium test case recorded by Selenium IDE
Executable Selenium test case template
A generated executable Selenium test case

Fig. 4. An example of an executable Selenium test case generated by merging a raw Selenium test case and the executable test case template.

3.4 Cacti Adapter

Once the JMeter master starts the load testing, Cacti adapter will periodically record the footprints of the target system through invoking Cacti every 10 seconds over SNMP protocol. The monitored attributes include usage information of CPU, memory, disk I/O and network traffic. While the JMeter master completes the load testing, Cacti adapter will stop recording the system footprint.

In order to generate the system footprints report, Cacti is extended to provide an HTTP service for extracting the system footprints during the period from the start time to the end time by modifying two Cacti programs Cacti/graph_view.php and Cacti/lib/timespan_settings.php. In Cacti/graph_view.php, we added code to make host_id to null as a default value for guest users. Hence, a guest user is not allowed to see any host's system footprints on it. Additionally, we modified Cacti/lib/timespan_settings.php to let Cacti support GET method for accessing recorded system footprint. The inputs of the extended system footprints HTTP service include a host ID and start/end time. The output of the service is a GUI report showing the footprints during the time period. An example of invoking the HTTP service is as follows: "http://cacti_server_ip/cacti/graph_view.php?host_id=9&starttime=2014-12-21%25:28&endtime=2014-12-21%35:41".

3.5 Xdebug Adapter

Xdebug is a PHP extension which provides debugging and profiling capabilities for PHP programs. It can record computation time of PHP functions while receiving a HTTP request. The time tracking logs for HTTP requests are stored in a log directory. Before Xdebug adapter accessing log files, the Xdebug log directory in the target system should be set to an online accessible web directory in order to have Xdebug adapter be able to access the Xdebug log files over HTTP protocol. When running a load testing, Xdebug will record the computation time of the invoked PHP programs, and the computation time will be stored into different log files for different requests sent from the JMeter slaves.

Because there will be a large number of log files generated during a load test, Xdebug adapter is going to select a representative log as a time tracking report for each PHP program. For the selection, Xdebug adapter will retrieve all of the time tracking logs generated on the target system during the period from the start to the end time of a load testing, and identify the time tracking log with the longest computation time as a time tracking report. The report can provide the tester an insight on execution time of each PHP function under a load test for further program performance turning.

3.6 Test Report Generator

Test report generator generates a compiled test report as an email containing information of the load testing results, cross-browser testing results, system footprints and time tracking report. It will also send the compiled test report to the tester through email. At first, the test report generator needs to configure the SMTP properties for connecting to a mail server. The mail server has to support SMTP protocol for sending emails. Subsequently, the adapter will generate an email with the content including the information of the load/cross-browser testing reports, the system footprints and the time tracking report. If the report cannot be generated successfully, the adapter will send an error notification email to the system administrator and the tester. Otherwise, the generated email will be sent to the tester. While the tester receives the email, the tester can click on the links in the email to view the test results.

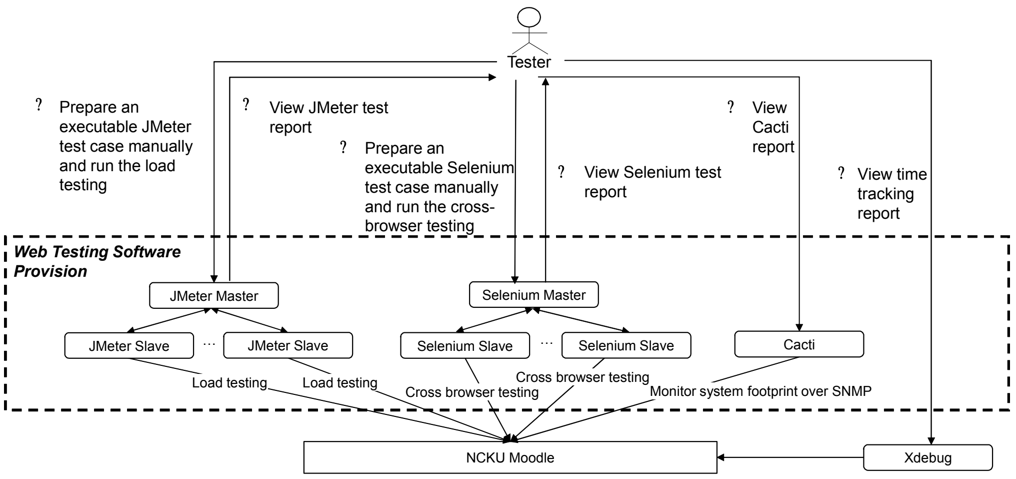
4. EXPERIMENTAL EVALUATION

This section presents the experimental evaluations of the proposed system. We designed the experiments to answer the following research question: *Does the proposed system reduce a tester's effort on conducting load testing and cross-browser testing in comparison with a conventional approach?* This research question is addressed by conducting an experiment with a real world web application. The following sections include the design of the experiment, the experiment, and the experimental results.

4.1 Design of the Experiment

In the experiment, we selected a real world web application system, NCKU Moodle, as the target system. JMeter/Selenium masters and slaves, Cacti, and Jenkins were deployed on virtual machines. Xdebug was deployed on the target system. Through a conventional approach (see Fig. 5), a tester has to prepare an executable JMeter test case or an executable Selenium test case manually according to a recorded raw test case. After the tests are complete, the tester has to open the JMeter/Selenium test reports. In addition, the tester has to open the Cacti and view the time tracking report through manually identifying the time period of the load testing. The tester also has to manually identify the time tracking log file of the longest execution time. Through the proposed system (see Fig. 6), the tester has to send a recorded raw JMeter test case or a raw Selenium test case to a particular email address and receive a compiled test report through email.

In the experiment environment, thirteen virtual machines were used for the system deployment. One VM is for hosting a Jenkins server. One VM is for running a JMeter master. Four VMs are for running JMeter slaves. One VM is for running a Selenium master. Five VMs are for installing Windows 7, 8.1, Linux (Ubuntu), OS X, and Android simulator (Bluestack). One VM is for running Cacti. All virtual machines are with the same hardware specification of 2.1 GHz Intel Xeon E5-2620v2 CPU (2 Core) and 16 GB memory.


Tester
? Prepare an executable JMeter test case manually and run the load testing
? View JMeter test report
? Prepare an executable Selenium test case manually and run the cross-browser testing
? View Selenium test report
? View Cacti report
? View time tracking report
Web Testing Software Provision
JMeter Master
JMeter Slave
JMeter Slave
Selenium Master
Selenium Slave
Selenium Slave
Cacti
Load testing
Load testing
Cross browser testing
Cross browser testing
Monitor system footprint over SNMP
NCKU Moodle
Xdebug
Fig. 5. Operational concept of conducting web testing with the conventional approach.

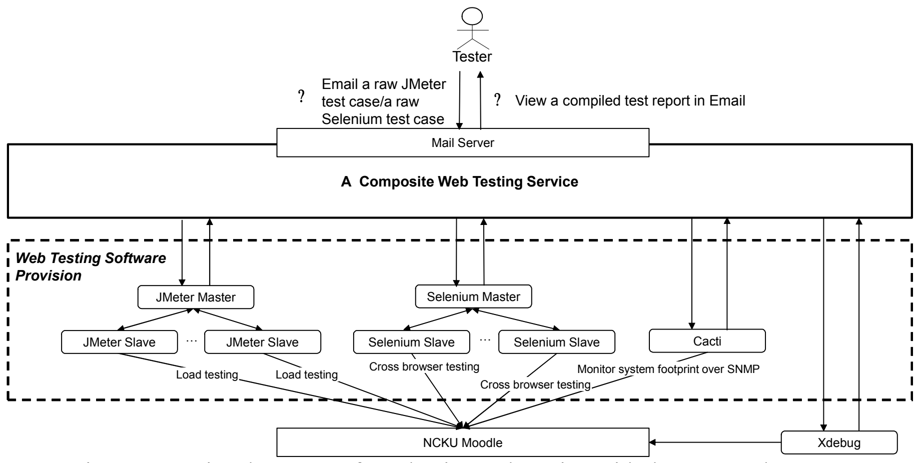


Fig. 6. Operational concept of conducting web testing with the proposed system.

4.2 Experiment

In the experiment, we randomly recorded 10 JMeter raw test cases and 10 Selenium raw test cases for testing NCKU Moodle, and three testers were involved for conducting load testing and cross-browser testing with the recorded raw test cases. Before the three testers started conducting the testing with the conventional approach, they were well trained to be familiar with using the related software, writing executable test cases, generating Cacti reports and specifying time tracking reports.

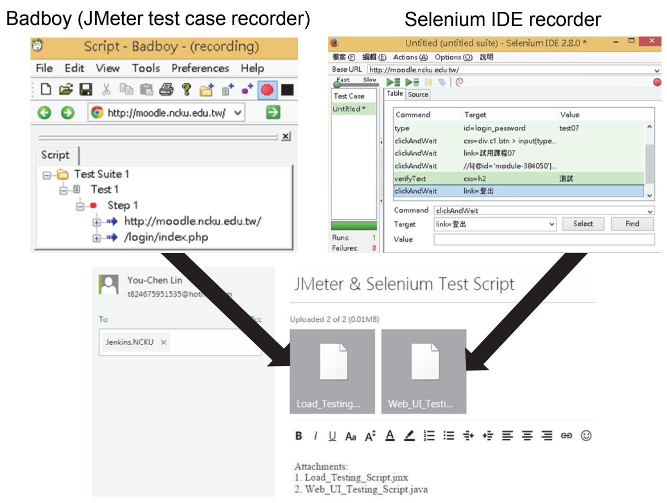
Fig. 7 shows the snapshot of recording a JMeter test case and a Selenium test case and requesting the test cases for execution through email with the proposed system. A tester can attach a JMeter test case file and a Selenium test case file to an email, and send the email to a particular email address of the system without filling out the email title and content. Fig. 8 shows the snapshot of receiving a compiled test report through email. The tester can open the load testing and cross-browser testing reports through clicking the links in the email. A new browser window will be automatically opened and redirected to the Jenkins server for showing the reports. In the same way, the system footprints of the load testing is shown through opening an automatically specified Cacti web page. The time tracking report is opened via Webgrind software.

4.3 Experimental Result

The average time spent by a user on completing all steps for running a test case is defined as follows.

Definition 1: Let $c_1, \ldots, c_n$ be test cases, $u_1, \ldots, u_p$ be users, and $s_1, \ldots, s_m$ be steps need to be completed for running a test case. t_{c_k,u_j,s_i} denotes the time user u_j spent on doing step s_i for test case c_k. The average time spent by a user on completing all steps for running test case c_k is calculated as

$$\overline{t}_{c_k} = \frac{1}{p}\sum_{j=1}^{p}\left(\sum_{i=1}^{m} t_{c_k,u_j,s_i}\right). \tag{1}$$


Badboy (JMeter test case recorder)
Selenium IDE recorder
Script - Badboy - (recording)
File Edit View Tools Preferences Help
http://moodle.ncku.edu.tw/
Script
Test Suite 1
Test 1
Step 1
http://moodle.ncku.edu.tw/
/login/index.php
Untitled (untitled suite) - Selenium IDE 2.8.0 *
Base URL http://moodle.ncku.edu.tw/
Test Case
Untitled *
Table Source
Runs: 1
Failures: 0
Select
Find
You-Chen Lin
To
Jenkins.NCKU
Uploaded 2 of 2 (0.01MB)
Load_Testing...
Web_UI_Testi...
Sending JMeter/Selenium raw test cases as attachment through email

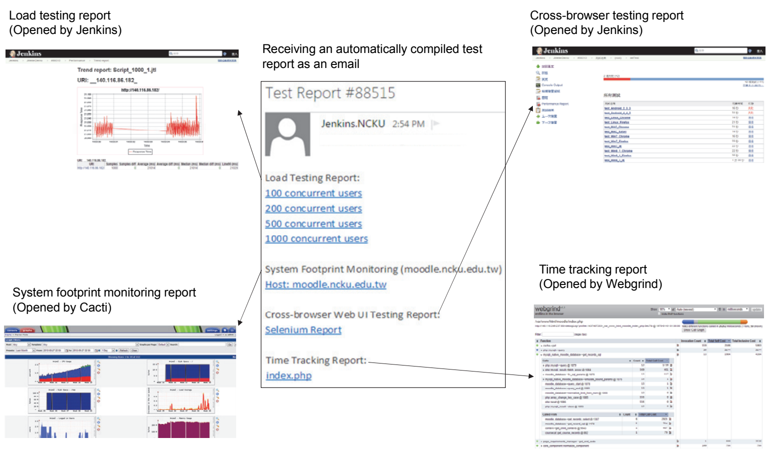
Command	Target	Value
type	id=login_password	test07
clickAndWait	css=div.c1.btn > input[type...	
clickAndWait	link=試用課程07	
clickAndWait	//li[@id='module-384050']...	
verifyText	css=h2	測試
clickAndWait	link=登出	

Command clickAndWait
Target link=登出
Value

JMeter & Selenium Test Script

Attachments:
1. Load_Testing_Script.jmx
2. Web_UI_Testing_Script.java

Fig. 7. Snapshot of recording JMeter/Selenium test cases and sending the test cases through email.


Load testing report
(Opened by Jenkins)
Receiving an automatically compiled test report as an email
Cross-browser testing report
(Opened by Jenkins)
System footprint monitoring report
(Opened by Cacti)
Time tracking report
(Opened by Webgrind)

Test Report #88515

Jenkins.NCKU 2:54 PM

Load Testing Report:
100 concurrent users
200 concurrent users
500 concurrent users
1000 concurrent users

System Footprint Monitoring (moodle.ncku.edu.tw)
Host: moodle.ncku.edu.tw

Cross-browser Web UI Testing Report:
Selenium Report

Time Tracking Report:
index.php

Fig. 8. Snapshot of receiving an automatically compiled test report through email.

The average time spent by a user on completing all steps for a test case is calculated as

$$\bar{t} = \frac{1}{n}\sum_{k=1}^{n}\bar{t}_{c_k}. \tag{2}$$

In the experiment, k is set to 10 and p is set to 3. For the conventional approach, 8 steps are involved for conducting a load test with a test case: (1) open Apache JMeter application, import the test case file, and then configure the test case settings; (2) modify the test case for simulating 100 concurrent users; (3) modify the test case for simulating 200 concurrent users; (4) modify the test case for simulating 500 concurrent users; (5) modify the test case for simulating 1000 concurrent users; (6) open the load test reports;

(7) open the system footprints report, then adjust the time section according to the load testing period; and (8) find out the time tracking log with the longest response time. The average time spent by a user on completing all steps for a JMeter test case through the conventional approach is denoted as $\overline{t}_{trad}^{load}$.

As for the cross-browser testing experiment with the conventional approach, four steps are involved with a test case: (1) create a Java project with initial settings; (2) import the cross-browser template file; (3) modify the Selenium commands in the raw test case; (4) copy the testing commands from the raw Selenium test case and paste them to every unit test in the template file. The average time spent by a user on completing all steps for a Selenium test case through the conventional approach is denoted as $\overline{t}_{trad}^{crossbrowser}$.

As for the load testing and cross-browser testing experiments with the proposed system, two steps are involved with a test case: (1) attach the test case to an email, and send out the email; (2) receive the mail containing the test reports and click links to open the reports. The average time spent by a user on completing all steps for a JMeter and a Selenium test case through the proposed system are denoted as $\overline{t}_{proposed}^{load}$ and $\overline{t}_{proposed}^{crossbrowser}$, respectively.

Fig. 9 shows the average time spent on completing each step of each test case through the conventional approach for load testing. With the conventional approach, a tester spent most of the time on completing steps 1, 7 and 8. For adjusting the time section in step 7, a user had to manually select the start and end time through operating panels, which also requires the tester to manually record the start and end time during load testing. For finding out the time tracking log with the longest response time, a tester had to drag and drop all of the tracking logs into an Xdebug log viewer, WinCacheGrind, and then view the time log in every log to identify the log with the longest response time. The average time $\overline{t}_{trad}^{load}$ of the conventional approach is 216.0. On the other hand, the average time $\overline{t}_{proposed}^{load}$ of the proposed system is 31.4. We conducted pair-sample *t*-test to know whether the difference between the average values of the two approaches is significant. The *t*-value for comparing the average values of $\overline{t}_{trad}^{load}$ and $\overline{t}_{proposed}^{load}$ is 101.4. The *t*-value is greater than 2.262 with 95% confidence level and 9 degrees of freedom, which means the difference between the average time of proposed system and the conventional approach is significant. Furthermore, the average time spent by a user on conducting a load testing with a test case through the proposed system is (216–31.4)/216 = 85.5% less than the one through the conventional approach.

Fig. 10 shows the average time spent on completing each step of each test case through the conventional approach for cross-browser testing. With the conventional approach, a tester spent most of the time on completing step 3. For modifying the Selenium commands in the raw test case in step 3, a user had to manually modify the raw test case by creating a number of new methods for different types of platforms and browsers and fill out the methods with the commands. The average time $\overline{t}_{trad}^{crossbrowser}$ of the conventional approach is 588.7, and the average time $\overline{t}_{proposed}^{crossbrowser}$ of the proposed system is 19.4. The *t*-value for comparing the average values of $\overline{t}_{trad}^{crossbroswer}$ and $\overline{t}_{proposed}^{crossbrowser}$ is 153.1. The *t*-value is also greater than 2.262 with 95% confidence level and 9 degrees of freedom, which means the difference between the average time of proposed system and the conventional approach is also significant. Furthermore, the average time spent by a user on conducting a cross-browser testing through the proposed system is (588.7–19.4)/588.7= 96.7% less than the one through the conventional approach.

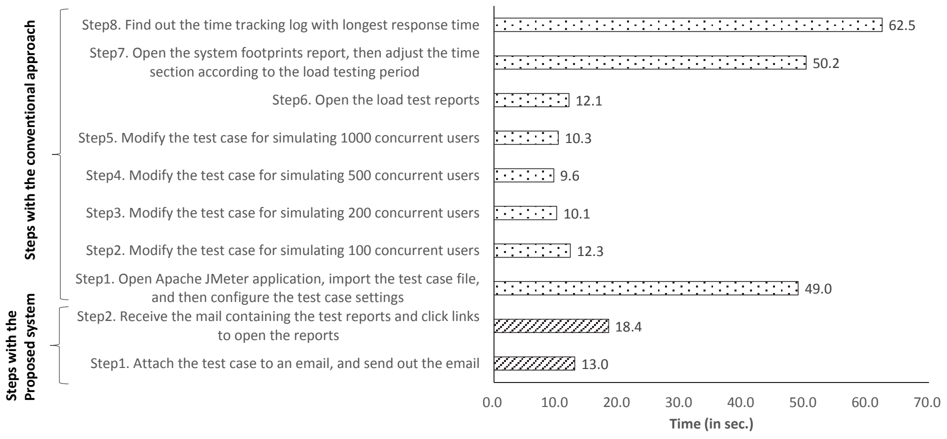


Fig. 9. Comparing the proposed system with the conventional approach on the average time spent on completing each step for load testing.

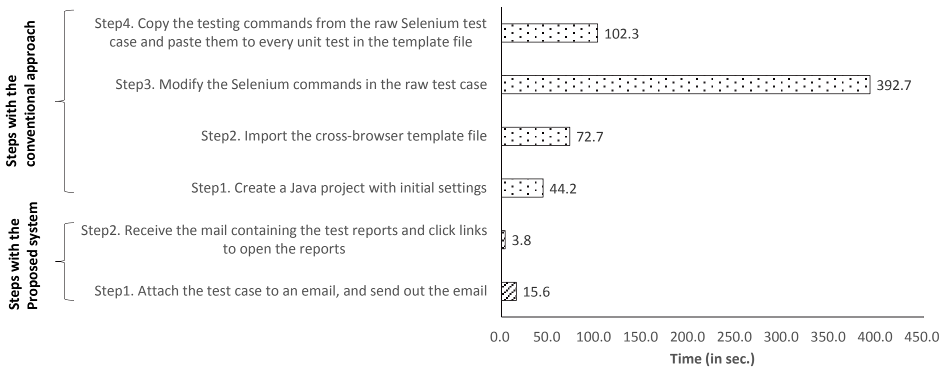


Fig. 10. Comparing the proposed system with the conventional approach on the average time spent on completing each step for cross-browser testing.

5. CONCLUSION

The growing number of heterogeneous web application testing software is driving the need for systems to assist testers in the use of these software. This paper proposes a system for the composition and delivery of heterogeneous web application testing tools aimed at reducing the effort associated with load testing and cross-browser testing. The proposed system integrates six state-of-the-art web application testing tools based on a continuous integration framework for the delivery of a composite web testing service via email.

The benefits of this work are twofold: (1) Inputs and outputs of the heterogeneous web testing software can be automatically converted without the intervention of testers; and (2) Testers are able to initiate the testing process simply by sending an email to the system with a raw test case attached and an email is automatically returned by the system with a compiled test report, including the results of load testing, cross-browser testing, system footprints, and time tracking results. The proposed system has been experimentally evaluated using a real-world web application. Our experimental results demonstrate that the average time required with the proposed system for load testing is 85.5% less than that required when using a conventional method. Furthermore, the time required for cross-browser testing can be reduced by 96.7%.

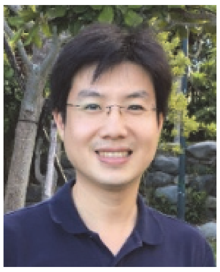
Our main objectives for future work include the following: (1) to support the compilation of test reports for multiple raw JMeter/Selenium test cases as attachment in an email; (2) to integrate the proposed system with additional software for the management of test cases and test reports; (3) to protect the system from malicious attacks and enhance the mechanism for the authorization of access rights to the testing resources; and (4) to develop additional adapters for the integration of additional software and design further composite web testing services for a range of testing scenarios.

REFERENCES

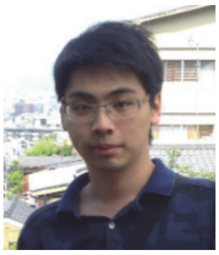
1. B. Floss and S. Tilley, "Software testing as a service: an academic research perspective," in *Proceedings of the 7th IEEE International Symposium on Service Oriented System Engineering*, 2013, pp. 421-424.
2. K. Inçki, I. Ari, and H. Sozer, "A survey of software testing in the cloud," in *Proceedings of the 6th IEEE International Conference on Software Security and Reliability Companion*, 2012, pp. 18-23.
3. J. Gao, X. Bai, W.-T. Tsai, and T. Uehara, "Testing as a service (TaaS) on clouds," in *Proceedings of the 7th IEEE International Symposium on Service Oriented System Engineering*, 2013, pp. 212-223.
4. L. Riungu-Kalliosaari, O. Taipale, K. Smolander, and I. Richardsona, "Adoption and use of cloud-based testing in practice," *Software Quality Journal*, Vol. 24, 2016, pp. 337-364.
5. C. Tao and J. Gao, "On building a cloud-based mobile testing infrastructure service system," *Journal of Systems and Software*, Vol. 124, 2017, pp. 39-55.
6. SeleniumHQ.org, "SeleniumHQ Browser Automation," http://www.seleniumhq.org.
7. J. Lee, S.-J. Lee, and P.-F. Wang, "A framework for composing SOAP, non-SOAP and non-web services," *IEEE Transactions on Services Computing*, Vol. 8, 2015, pp. 240-250.
8. F. Curbera, M. Duftler, R. Khalaf, and D. Lovell, "Bite: workflow composition for the web," in *Proceedings of the 5th International Conference on Service-Oriented Computing*, 2007, pp. 94-106.
9. S. Dustdar and W. Schreiner, "A survey on web services composition," *International Journal of Web and Grid Services*, Vol. 1, 2005, pp. 1-30.
10. J. Niemoller, K. Vandikas, R. Levenshteyn, D. Schleicher, and F. Leymann, "Towards a service composition language for heterogeneous service environments," in *Proceedings of the 15th IEEE International Conference on Intelligence in Next Generation Networks*, 2011, pp. 121-126.
11. K. He, "Integration and orchestration of heterogeneous services," in *Proceedings of IEEE Joint Conferences Pervasive Computing*, 2009, pp. 467-470.

12. S.-J. Hsieh, G.-H. Luo, S.-M. Yuan, and H.-W. Chen, "A flexible public cloud based testing service for heterogeneous testing targets," in *Proceedings of the 16th Asia-Pacific on Network Operations and Management Symposium*, 2014, pp. 1-3.
13. M. Kamra and R. Manna, "Performance of cloud-based scalability and load with an automation testing tool in virtual world," in *Proceedings of the 8th IEEE World Congress on Services*, 2012, pp. 57-64.
14. X. Wang, B. Zhou, and W. Li, "Model based load testing of web applications," in *Proceedings of International Symposium on Parallel and Distributed Processing with Applications*, 2010, pp. 483-490.
15. O. Hamed and N. Kafri, "Performance testing for web based application architectures (.NET vs. Java EE)," in *Proceedings of the 1st International Conference on Networked Digital Technologies*, 2009, pp. 218-224.
16. Q. Gao, W. Wang, G. Wu, X. Li, J. Wei, and H. Zhong, "Migrating load testing to the cloud: a case study," in *Proceedings of the 7th IEEE International Symposium on Service Oriented System Engineering*, 2013, pp. 429-434.
17. G. Candea, S. Bucur, and C. Zamfir, "Automated software testing as a service," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010, pp. 155-160.
18. A. Cervantes, "Exploring the use of a test automation framework," in *Proceedings of IEEE Aerospace Conference*, 2009, pp. 1-9.
19. T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, T. Hanawa, and M. Sato, "D-Cloud: design of a software testing environment for reliable distributed systems using cloud computing technology," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster*, *Cloud and Grid Computing*, 2010, pp. 631-636.
20. A. Gambi, W. Hummer, and S. Dustdar, "Automated testing of cloud-based elastic systems with AUToCLES," in *Proceedings of IEEE/ACM 28th International Conference on Automated Software Engineering*, 2013, pp. 714-717.
21. V. Dallmeier, M. Burger, T. Orth, and A. Zeller, "WebMate: generating test cases for web 2.0," in *Proceedings of International Conference on Software Quality*, 2013, pp. 55-69.
22. M. Yan, H. Sun, X. Wang, and X. Liu, "WS-TaaS: a testing as a service platform for web service load testing," in *Proceedings of the 18th IEEE International Conference on Parallel and Distributed Systems*, 2012, pp. 456-463.
23. M. Yan, H. Sun, X. Wang, and X. Liu, "Building a TaaS platform for web service load testing," in *Proceedings of IEEE International Conference on Cluster Computing*, 2012, pp. 576-579.
24. D. Li and C.-H. Yang, "A test platform for evaluation of web service composition algorithm," in *Proceedings of the 3rd International Symposium on Intelligent Information Technology and Security Informatics*, 2010, pp. 302-305.
25. W.-T. Tsai, P. Zhong, J. Balasooriya, Y. Chen, X. Bai, and J. Elston, "An approach for service composition and testing for cloud computing," in *Proceedings of the 10th International Symposium on Autonomous Decentralized Systems*, 2011, pp. 631-636.
26. M. Felderer, P. Zech, F. Fiedler, and R. Breu, "A tool-based methodology for system testing of service-oriented systems," in *Proceedings of the 2nd International Conference on Advances in System Testing and Validation Lifecycle*, 2010, pp. 108-113.

27. I. Bluemke, M. Kurek, and M. Purwin, "Tool for automatic testing of web services," in *Proceedings of Federated Conference on Computer Science and Information Systems*, 2014, pp. 1553-1558.



Shin-Jie Lee (李信杰) is an Associate Professor in Computer and Network Center at National Cheng Kung University (NCKU) in Taiwan and holds joint appointments from Department of Computer Science and Information Engineering at NCKU. His current research interests include software engineering and service-oriented computing. He received his Ph.D. degree in Computer Science and Information Engineering from National Central University in Taiwan in 2007.



You-Chen Lin (林宥辰) received his B.S. degree in Department of Computer Science and Information Engineering from National University of Tainan in Taiwan in 2013. His research interests include information and mechatronics integration systems, manufacturing systems, object-oriented programming, web service, IaaS could infrastructure.

Kun-Hui Lin (林昆輝) received his B.S. degree in Department of Computer Science and Information Engineering from National Cheng Kung University in Taiwan in 2013. His research interests include information and mechatronics integration systems, manufacturing systems, object-oriented programming, web service, PaaS could platform.

Jie-Lin You (游傑麟) is a graduate student in Department of Computer Science and Information Engineering at National Cheng Kung University in Taiwan.