A Software Testing Service Supporting Long-Lived and Asynchronous Testing Jobs

Shueh-Cheng Hu and I-Ching Chen, Members, IACSIT

Abstract — Testing as a service (TaaS) provides features like easiness of use, high availability, low installation and maintenance cost, thus can aid programmers and white-box testers in improving software quality. Automatic test data generation being embedded in a TaaS must consume significant amount of time and computer resources while dealing with mid- or large-scale programs. However, allowing large volume of long-lived test sessions in a TaaS will incur adverse impact on the scalability of the TaaS and its hosting servers. To enable a TaaS supports time-consuming testing jobs and keeps its scalability simultaneously; a message queuing system could be adopted for holding un-processed test jobs temporarily. This article describes the corresponding design rationale, implementation details of a testing as a service with message queuing system and discusses its advantages.

Index Terms—Java path finder, message queuing system, software quality assurance, testing as a service.

I. INTRODUCTION

To assure the quality of software, testing and/or verification are necessary to be performed before delivering software to users. Typically, software testing work comprises test data preparation, test script execution, and results collection and analysis, all of these are labor-intensive and time-consuming. Consequently, testing consumes considerable portion of resources including budget, in most software development projects. In general, the cost of testing works exceeds more than forty percent of the total budget in a software development project, according to a report from a leading software vendor [1].

Unit-testing, confirms the correctness of each software component before integrating related components as larger structures such as packages, sub-systems, or systems. In practical, programmers and white-box testers rely on unit-testing to assure quality of software components that they developed. Prior study indicated that fixing a software bug in its post-released stage will cost 10 times or even more than fixing the same bug in earlier stage such as unit testing [1]. Obviously, unit-testing plays a significant role in cost-efficient quality assurance of software.

Usually, programmers need to perform adequate test on each software component. However, due to lack of experience and tight schedule, it is a challenge for them to design and execute unit testing works that can meet schedule as well as is adequate enough based on a particular coverage criterion [2]. In consequence, programmers and white-box testers cherish a utility that can aid them in performing unit testing works adequately but efficiently, as well as be easy to use and highly available to them.

In view of its significance in the area of software quality assurance, provisioning an easy to use and highly-available unit test service is critical and valuable. There are volume of prior studies and practical software tools available for assuring software quality, such as testing as a service (TaaS) in the age of cloud computing. Inevitably, each TaaS will encounter some submitted programs that are large enough to consume significant amount of time and computer resources to complete. In consequence, long-lived test sessions will be created and exist in that TaaS and the hosting servers, which adversely affect the scalability of the testing service.

Up to date, there are rare research works concentrating on the handling of long-lived testing jobs in a TaaS. Accordingly, this article presents a testing service that not only to offer automatic test data generation and test script execution for better efficiency, but also being able to handle ever-changing volume of short- and long-lived testing jobs. The anticipated contribution of the present service is to assure quality of software in a more efficiently and effectively way.

II. PRIOR STUDIES AND TECHNIQUES REVIEW

Before presenting the software testing service that aims to support long-lived and asynchronous software quality assurance jobs, the prior studies and relevant techniques including automatic software testing, testing as a service, and message queuing software are reviewed.

A. Automatic Software Testing

Among multiple techniques, testing is the most popular way to assure quality of software. According to their purposes, there are different types of testing methods: unit testing, integrated testing, usability testing, performance testing, etc [3]. Among others, unit testing is the most frequently used method. The main function of unit testing is to make sure each individual component works correct, or ensure that component meets its functional requirements and behaves as intended [4]. No matter which (procedural, object-oriented, or other) programming technique is used, the quality of software components is a crucial factor determining the quality of an aggregated software system. Obviously, the role of unit testing is analogous to the quality control procedure for checking quality of electronic
components, which will be assembled as a computer or consumer electronic product for end users.

There have been 3 major challenges for conducting an adequate but cost-efficient software testing work [5]. First, the required computing resources and working hours grow exponentially while the time complexity of software increases. As a result, limited resources under many circumstances is inadequate for testing a mid- or large-scale software. Second, testing work is difficult to cover all possible execution scenarios under limited budget and tight schedule, especially when the component is large and/or testing needs manual works. Thus, software bugs with potential risks might likely hide in untested segments, which will compromise the software quality. Third, constraints exist in utilizing testing resources including test beds and tools, i.e., significant amount of preparation is necessary before conducting testing, so people cannot perform testing works frequently and in early stages of software lifecycle. Obviously, all these challenges hinder a thorough but efficient testing work, which leads to an inferior software quality.

To address these challenges, automated test data generation and test script execution [6]-[8], software testing services [9], [10] were proposed accordingly.

B. Testing as a Service

As one of the service model in cloud computing, software as a service (SaaS) [11]-[13] should be the one which most end users of software can benefit from. SaaS reshapes the paradigm through which software deliver functions to users; unlike traditional software that need to pre-acquired and installed on particular host(s), SaaS deliver user functions that were wrapped in Web-based interfaces through the Internet, or more specifically, the Web service techniques [14]. Users of SaaS can access the latest updated functions of a software without installing and refreshing the required software by themselves, the software are maintained and controlled by software providers on remote sites.

In a SaaS model, functionalities of software are delivered to users through the Internet and HTTP; users do not need to handle executable files and configuration data, instead, these files are hosted in the cloud. Basically, users can consume SaaS via a Web client (browser) that might run on diverse form factors such as smart phones, thin clients, tablet PCs, desktops, and so on. Besides offering a fairer pricing scheme that charge users based on actual usage, the SaaS model accelerated functions and data revision, from end users’ perspectives. This feature is very critical to modern e-business and e-commerce platforms where people are always eager for new functionalities for more efficient operations and better competitiveness, but are reluctant to maintain installed various software.

Testing as a service (TaaS) [10], [15], [16], one particular sub-category of SaaS applications, aims to provide software developers and white-box testers an easy-to-use and cost-efficient testing utility in a form of service. Rationally, TaaS should be deployed in a cloud infrastructure to leverage its advantages such as scalability, just like other SaaS applications. Consequently, TaaS inheriting merits of SaaS, to certain extent, can solve the difficulties that adopters of software testing services are encountering. First, elastic allocation of computing power and storage offered by cloud infrastructures can ease administrative works of an software testing environment; i.e., users only need to focus on their core business: testing the target software, and do not worry about the allocation of required resources.

Second, service-based delivery of testing functionality minimizes the preparation effort before conducting software testing, which further makes testing become a ubiquitous and highly available function. As a result, with TaaS tools, programmers and white-box testers have better chance to identify and correct errors in early stage, which will significantly avoid the higher cost of fixing bugs in late stages of software lifecycle.

Other prime advantage of TaaS toward establishing an elastic software testing environment is that TaaS create a new and fair payment scheme: pay-per-use, which means that users request testing functionality, in form of service, from service provider whenever they actually need to use these functions. After requesting and consuming the services, users need to pay for the particular usage only, and the one-time payment usually is only a fraction of the full-year fee for purchasing or licensing a software testing tool.

C. Message Queuing System

Typically, a message queuing software is used to coordinate a number of distributed or loosely-coupled software systems, which might run on different operating systems or different hosts [17]. To reflect this main purpose, it is also called message-oriented middleware (MOM) [18].

Besides being able to integrate distributed software systems in a heterogeneous set of environments, it’s another merit is supporting asynchronous message passing, which means a message’s producing and consuming activities can be handled by two clients respectively and the activities happen in two different stages. Supporting asynchronous communication mode is critical in distributed software environments because there are many scenarios in which synchronous communication between message producer and consumer is improper. For example, those circumstances in which processing a message and making the response need large but unpredictable amount of time, it is unreasonable for a message producer to wait for responses and block its subsequent operations.

To support reliable and dual modes (synchronous and asynchronous) of communication, a typical message queuing system usually include the following components: a message broker that manages clients’ messages, sending and storing of messages; message queue; queue persistence mechanism such as database system; the APIs for clients they need to produce or consume messages in the queuing system.

To create a common, more interoperable MOM for coupling Java-based software systems, the Java Message Service (JMS) [19] specification (essaging system with Java interface) was developed and there are many implementations based on it, such as ActiveMQ [20] from Apache software foundation, WebSphere MQ [21] (formerly MQ Series) from IBM, HornetQ [22] from the JBoss, etc. There are two operation models in the JMS: point-to-point
and publish/subscribe models. In the former one, each message is addressed to a specific queue, and the consuming clients retrieve messages from the queues established for holding their messages. The latter model supports clients to publish messages to a particular message topic, and then subscribers may register to receive messages that were published on a particular topic.

III. SYSTEM ARCHITECTURE AND DESIGN RATIONALES

Three major purposes of the present service were identified as follows: first of all, users can assure Java programs' correctness in an easy and efficient way. Besides, both project manager and programmers can use this testing service to monitor and measure both short- and long-lived testing jobs in a software development project. Furthermore, the testing service should be scalable, thus a cost-efficient TaaS could be realized.

A. System Architecture

Keeping these purposes in mind, the system architecture of the present service is shown in Fig. 1. As Fig. 1 illustrates from a static perspective, there should be six major function modules in the present service system:

1) A HTTP and application server that accepts clients' requests, executes a particular command such as forming a test job and put that job into a queue, and delivers results as responses.

2) A Java compiler that converts uploaded Java source codes into byte codes, which is the format could be analyzed by automatic test data generator later.

3) A message queuing sub-system that can accommodate requested but un-processed test jobs from users temporarily before the next processing steps could be executed subsequently.

4) A test data generation module that can generate test data automatically based on given input, Java byte code in this project. The results of applying these test data could be asserted by programmers or white-box testers later.

5) A test script execution modules, i.e., a test engine, which executes test scripts to exercise feasible paths of the tested program. The results of executing the test scripts will be formatted as a test report in HTML and will be available to users with adequate privileges.

6) A service wrapper that encapsulate all other function modules and data modules with a restful service-based interface (e.g., API), so that clients who might be users or intelligent agents can perform testing tasks through this interface.

B. Design Rationales

The integration of a message queuing system into the present testing service will be justified first. Apparently, for a general TaaS provider, there must be some test jobs that need to consume large amount of time and memory space before completion. That is because the number of feasible paths in a program grows exponentially with the number of decision points in that program. For example, there will be over 1 billion (about 2^30) of possible execution paths in a moderate-sized program with 30 non-nested two-branch decision points. Considering the test data generation module need to identify each path and derive the corresponding data for exercising that path, consequently, for each mid- or large-scale program, path analysis and test data generation will take a long time to complete, not to mention the subsequent tasks such as test script execution, testing results assertion, and reporting. That means general users who request testing a mid- or large-scale program via a shared public service should not expect receive their testing reports within a short period of time, say, 2~3 minutes or less.

<table>
<thead>
<tr>
<th>Resource identifier</th>
<th>HTTP method</th>
<th>Description of operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.taas.com/classes">http://www.taas.com/classes</a></td>
<td>GET</td>
<td>Retrieve all class records</td>
</tr>
<tr>
<td><a href="http://www.taas.com/username/classes">http://www.taas.com/username/classes</a></td>
<td>GET</td>
<td>Retrieve all class records owned by a specific user</td>
</tr>
<tr>
<td><a href="http://www.taas.com/username/classes/cn=name_of_new_class">http://www.taas.com/username/classes/cn=name_of_new_class</a></td>
<td>POST</td>
<td>Create a new class record for a specific user</td>
</tr>
<tr>
<td><a href="http://www.taas.com/username/classes/class_name">http://www.taas.com/username/classes/class_name</a></td>
<td>GET</td>
<td>Retrieve a particular class record owned by a specific user</td>
</tr>
<tr>
<td><a href="http://www.taas.com/username/classes/class_name">http://www.taas.com/username/classes/class_name</a></td>
<td>PUT</td>
<td>Update a particular class record owned by a specific user</td>
</tr>
<tr>
<td><a href="http://www.taas.com/username/classes/class_name">http://www.taas.com/username/classes/class_name</a></td>
<td>DELETE</td>
<td>Delete a particular class record owned by a specific user</td>
</tr>
</tbody>
</table>

However, the HTTP enabling provision of the testing service itself is synchronous one, which means a test session with a mid- or large-scale program involved will be non-responsive from users' perspective. Besides the latency adversely affect users experiences, long-lived test sessions definitely consume large amount of resources including.
memory spaces in computers hosting the services. Taking its time and resource consumption issues into account, long-lived sessions will significantly decrease the scalability of the servers as well as the testing service being provisioned based on them.

Obviously, it is rational to place user-requested testing jobs to a message queueing system to make test sessions in HTTP servers short-lived, i.e., synchronous. In other words, users can get testing job identification immediately after submitting a test job to the TaaS, but they need to wait for a while before being able to retrieve or receive the results showing in test reports, which might take minutes or even hours, depends on complexity of the submitted program and the work load of the TaaS servers.

Due to the overhead of volume message processing, the SOAP-based Web services became bulky to sites with intensive traffic. To address this issue, the REST-styled service presented by Roy Fielding in his dissertation [23] is a promising approach for architecting software testing services that might serve vast amount of user requests in particular period of time [24]. The restful style [25] Web service emerged as a lightweight alternative for realizing the concept of software as a service [26], which is one of the major service delivery models in cloud computing environment.

Regarding the service wrapper, it is actually another restful Web application, its main purpose is hiding details of all other functional and data modules with a restful styled interface, so that clients can perform testing tasks via diverse types of devices supporting HTTP-client, which leads to a more accessible testing environment and reduces software defects accordingly.

IV. PROOF OF CONCEPTS

The proof of concepts implementation based on the aforementioned design rationales are detailed in this section. There are five major parts in this testing service system: a HTTP and application server, an ActiveMQ message queuing system, a JPF-symbolic module, a Junit and Ant-based test case execution engine, and a service wrapper for encapsulating core functions with restful styled interface.

A. Web and Application Server

Apache Tomcat was used as the Web and application server for the testing service. Java servlets serve as service provisioning programs that accept users request, form testing jobs and then send them to the message queue system. Each testing job will comprises two sessions: the synchronous one ends shortly along with a testing job identification and an asynchronous one that might live in the message queueing system or test data generation modules for a while.

B. ActiveMQ Messaging System

Apache ActiveMQ, an open-sourced and popular implementation of the JMS, was used as the message queuing system for the testing service. It accommodates un-processed testing jobs that might be time-consuming, thus makes HTTP client sessions short-lived or synchronously in this service system. The point-to-point model was applied in this particular implementation; the servlet receiving testing requests send jobs to the message queue, from which the following test data generation module retrieves the next un-processed testing job.

C. Java Path Finder and JPF-Symbolic

Symbolic Java path finder (JPF-symbolic) [27], [28], was used as the test data generator in the present system, which analyzes the submitted Java byte codes and then applies the symbolic execution technique to generate test data covering all execution paths of the tested program completely. The outcome of this module was tuned as the format of Junit [29], a package for unit-testing Java classes. JPF is an open-sourced library for automatically generating test data of Java programs (classes), it has being developed and maintained by the NASA. The symbolic execution technique has been widely adopted by many automatic test data generation tools since its introduction by James C. King as early in 1976 [30].

D. Junit and Ant Test Case Execution Engine

The Ant [31] and the corresponding commands, as a test engine, which executes test scripts in the form of Junit to exercise feasible paths of the tested program and assert the execution results. The results of executing the test scripts will be formatted as a test report in HTML and will be available to users with adequate privileges.

E. Restful Style API

To create a restful styled interface for functions in this service system, the interface must comply with the following constraints:

1) all types of resources will be organized in hierarchical style. For example, an user's class consists of several methods that in turn comprising different paths and test cases.

2) identifiers of all resources follow the common URI pattern, and thus could be represented hierarchically. For example, to identify a particular class (say, with class name "apple") within an user's all classes, we can use http://www.taas.com/username/classes/apple.

3) the four basic HTTP methods: POST, GET, PUT, and DELETE are uniformly used to map the CRUD (Create, Read, Update, Delete) operations on a resource, respectively. Accordingly, to create a new instance of the class record for a particular user, a POST request including the HTTP verb along with the request input data specified in a URL parameter would be sent to http://www.taas.com/username/classes.

Some resource identifiers of the system, the applied HTTP method, and the corresponding operation performed on the identified resource are extracted and listed in Table I. This article takes Java classes, i.e., software components in Java based system, as examples of resources in the unit testing system, because classes are generally considered to be one major kind of resources in a unit testing context.

On the client side of the proof-of-concept system, the AJAX was applied to send the four basic HTTP requests, as well as to receive and handle the response messages from the service provider. Typical restful applications allow that resources have different formats, e.g. plain text, XML, and
V. DISCUSSIONS AND CONCLUSIONS

This article presents a testing as a service for Java programmers and white-box testers, the purposes of this kind of service include making software testing tasks be easy to use and highly available, as well as supporting test jobs that might be long-lived due to high complexity of programs or heavy load of the service hosting server.

A. Anticipated Advantages

Comparing with conventional manual testing or non-service testing utilities, the anticipated advantages of the present testing service include:

1) from the viewpoint of software vendors, the time to market and the cost could be reduced because the conventionally time-consuming testing phase could be condensed via efficient and highly available testing utility, which applies automatic test data generation and test script execution. Typically, test data preparation and test script execution are labor intensive.

2) from the viewpoint of project manager, it is easy to maintain a certain level of software quality; because quality of unit testing tasks tend to be fluctuating if tasks are manually performed by disparate individuals who usually possess different levels of expertise, experience, and patience. In contrast, automatic test data generation and execution offered via the UTaaS will minimize the individual differences. Besides, consolidated test report and statistics from the service provide managers a manageable overview about all software components' status and the progress of unit testing works.

3) from the viewpoint of programmers or those who perform unit testing, a more accessible testing utility will make testing works more thorough. From temporal perspective, testing works could be conducted within a wider span of lifecycle with an easy-to-use and highly available service, which means higher chances of bug identification and correction. From spatial perspective, higher coverage of programs' execution paths could be achieved within the same period of time with the service due to higher efficiency of the automatic operations.

B. Future Works

To accommodate requests from massive clients and provide response within reasonable period of time, some issues regarding the performance and resource allocation need to be addressed further. These issues include load balancing and the parallelization of those operations including test data generation that need to apply symbolic execution and thus must consume volume of computing resources. Moreover, to make this kind of service more applicable, some function modules in the service need to be adaptive to software development projects using different programming languages.

REFERENCES


Shueh-Cheng Hu was born in Taichung city, Taiwan in 1965. He received both B.A. and M.S. degrees in computer engineering from National Chiao-Tung University, HsinChu, Taiwan, in 1987 and 1989, respectively. He received his Ph.D. degree in computer science from Texas A&M University, College Station, TX, in 2000.

He is an associative professor in the Department of Computer Science and Communication Engineering at Providence University, Taiwan. Prior to Providence University, he held various software system research and development positions at a number of companies and organizations including AT&T Lab, Taiwan Stock Exchange Corporation, etc. Dr. Hu has been pursuing research in the areas of Web technology, service-based software, e-learning, and e-commerce enabling technologies since 2004. He has published over 30 refereed papers in relevant journals and conferences, which majorly focus on the areas of software development, Web and cloud technologies, e-learning, and e-commerce.

Dr. Hu currently is a member of IEEE, ACM, IACSIT, and IEDRC. He also has served as reviewers for a number of international journals, and committee members of international conferences.

I-Ching Chen was born in Yuanlin county, Taiwan in 1973. She received her B.A. degree in international trade and business from Tunghai University, Taichung city, Taiwan in 1997, and M.S. degree in computer science from the same school in 2002. In 2011, she received her Ph.D. degree in information management from National Yunlin University of Science and Technology, Yuanlin, Taiwan.

She is a faculty member at Chung-Chou University of Science and Technology, where she is an assistant professor in the Department of Information Management. Prior to Chung-Chou University, she held various teaching and administrative positions at Tunghai University and other universities in central Taiwan. She has been pursuing research in the areas of management information system (MIS), e-commerce, and customer relationship management (CRM), and e-learning since 2005. She has published over 30 refereed papers in relevant journals and conferences, which majorly explore issues in the areas of MIS, CRM, e-commerce, e-learning, and Web-based software.

Dr. Chen currently is a member of IEDRC. She also has served as reviewers for a number of journals, and committee members of international conferences.