A General Framework for Testing Web-Based Applications

Saeed Abrishami, Mohsen Kahani
Computer Engineering Department, Ferdowsi University of Mashhad
s-abrishami@um.ac.ir r, kahani@um.ac.ir

Abstract

Software testing is a difficult task for web based applications due to their special features like multi-tier structure and emergence of new technologies (e.g. Ajax). In recent years, automatic testing of web based applications has been emerged as a promising technique to tackle the difficulties of testing these types of applications and several frameworks have been proposed for this purpose. But the most important problem of these frameworks is the lack of generality for different types of tests and programming environments. In this paper, we proposed a general framework for automatic testing of web based applications that covers all aspects of different types of testing, in an arbitrary web based application.

1. Introduction

Frequent use of the internet for crucial tasks, creates serious concern for the quality of web-based software systems. Web based system tends to change rapidly, due to emergence of new technologies (like Ajax) and the demands of users. In such a highly variable environment, manual testing of softwares is a hard and time consuming task, since automated software testing is an inevitable choice for testing the web-based software.

Software testing methods are traditionally divided into black box testing and white box testing. Black box testing treats the software as a black-box without any understanding of internal behavior. It aims to test the functionality according to the requirements. Thus, the tester inputs data and only sees the output from the test object. White box testing, however, is when the tester has access to the internal data structures, code, and algorithms. White box testing methods include creating tests to satisfy some code coverage criteria, and also can be used to evaluate the completeness of a test suite that was created with black box testing methods. In recent years the term grey box testing has come into common usage. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.

Several techniques have been proposed for the testing of web-based applications as both, research proposals and commercial tools. It is roughly possible to categorize such techniques into three groups[1]: (1)functional testing techniques, supporting requirement-base testing; (2) structural techniques, supporting some form of white box testing based upon the analysis and instrumentation of source code; and (3) model-based techniques, which exploit a navigation model of the application.

Different frameworks aim to construct an infrastructure for automatic testing of web-based applications. Sampth et. al. [2] proposed a framework that uses user session logs to generate test cases and then a replay tool sends these generated test cases to the server and collects the results. These results send to a test oracle that compares them to the expected results. This framework also uses a Coverage Analysis Tool measure the adequacy of the test suit, using statement and method coverage testing.

Zhu [3] proposed a framework for testing of web services. In this framework, each web service should be accompanied by a testing service. In addition to these testing services, testing tool vendors and companies have independent testing services to perform various kinds of test tasks like to generate test cases, to measure test adequacy, to

extract various types of diagrams from source code and so on. The trusted independent test services can call the testing services belong to a web service, to access the internal information (such as source code).

Chu et. al [4] presented a testing framework called FAST (Framework for Automating Statistics-based Testing) based on a method called statistical testing. Statistical testing techniques involve exercising a piece of software by supplying it with test data that are randomly drawn according to a single, unconditional probability distribution on the software's input domain. This distribution represents the best estimate of the operational frequency for the use for each input.

One of the most important parts of this framework is Automated Test Data Generator that is responsible for generating test cases. The approach adopted in this paper is specification of the input domain of a software by means of a SIAD (symbolic input attributed decomposition) tree which is a syntactic structure representing the input domain of a piece of software in a form that facilitates construction of random test data for producing random output for quality inspection.

In a similar manner, the specification of each product unit (output) is addressed by the SOAD (symbolic output attributed decomposition) tree. A SOAD tree can be used as a tool for describing the expected result which satisfies the user's requirement and as a basis for analyzing the product unit automatically, without a test oracle. The Quality Analysis module analyzes the product units and finds the "Defective Outputs". At the end, the defect rate has been computed using a binomial distribution.

MDWATP (Model Driven Web Application Testing Program) is a framework proposed by Li et. al. [5] for testing web application based on model driven testing. It uses 4 principal models based on UML2.0 standard. The first model is System Under Test (SUT) View that presents the model of the system being tested. So the navigation is a basic characteristic of web applications, they used a navigation model proposed by Lucca et. al. and also by Ricca and Tunella, named Web Application Navigation Model (WANM).

WANM depicts the navigation relations among the client pages, and hyperlinks and forms in each client page. Based-on the SUT View, test cases are automatically or semi-automatically generated. The generated test cases are described in the Test Case View, that is a model extends UML sequence diagram. Each test case is a sequence of client pages to be accessed. After that, the process and environment of test execution are modeled in the Test Execution View. In the test execution view, two kinds of models are defined: the test deployment model that extends the deployment diagram of UML2, and the test control model that extends the UML activity diagram.

After the execution engine automatically executes test cases based-on models described in the Test Execution View, Finally test results are represented in the Test Result View. Test result model is defined to save and present test results. And besides being shown in reports, test results would be associated with test case models.

But current frameworks have two major problems: (1) they concentrate on special aspects of testing and there is no general framework that contains all elements and types of testing for web-based applications, and (2) application developers don't like to share their internal information, models and source codes with others (including external application testers) and this makes white box testing so difficult or even impossible. In this paper we have proposed a framework that solves these two problems. This framework has been explained in section 2.

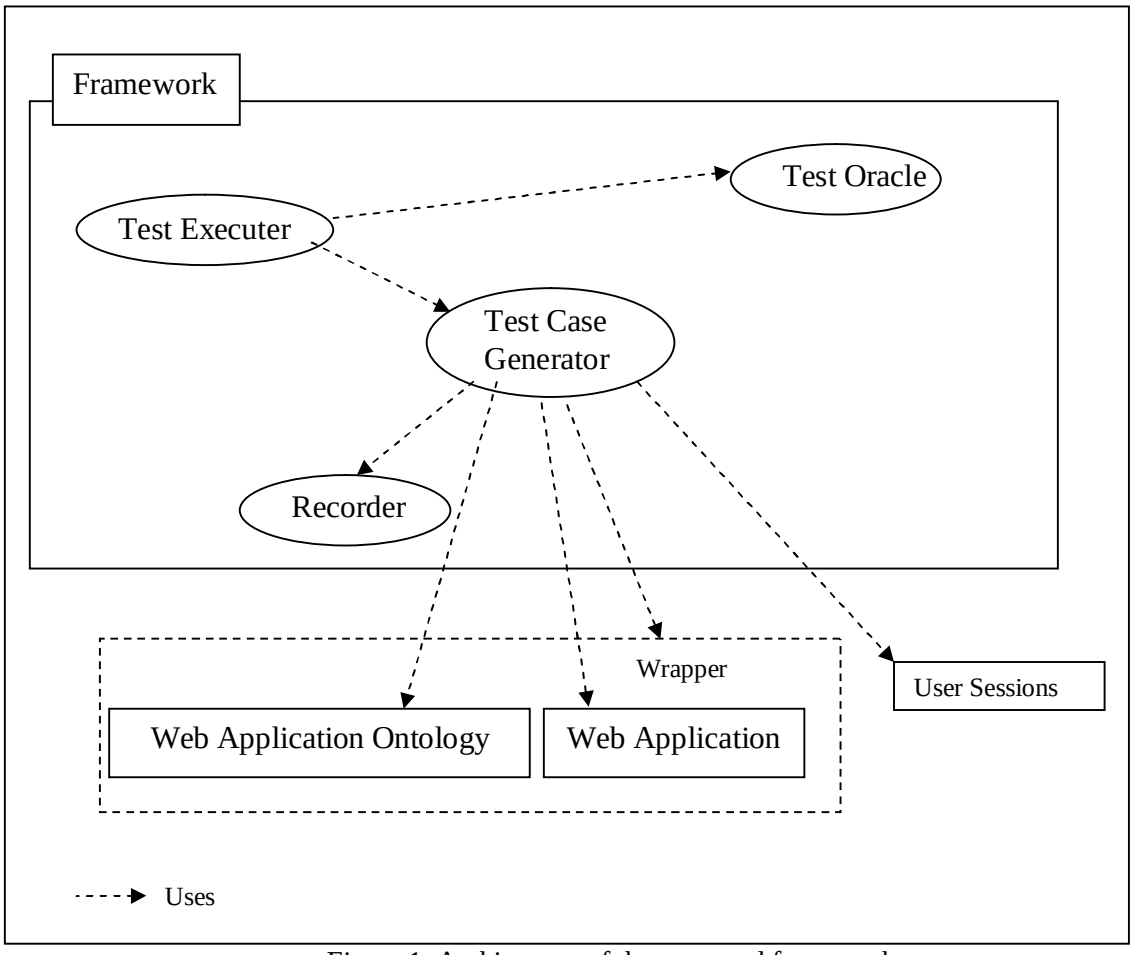

Figure 1. Architecture of the proposed framework

3. The General Framework

In this section we review our proposed framework for software testing. The general framework and its components have been shown in figure 1.

STOL language: As the components of the framework has to exchange information with each other (and with the external world), we require a common language for this purpose. We define STOL (Software Testing Ontology Language), an XML-based language with the required terms and relations (ontology) for software testing. It has been used for explaining program models, test case input ranges, etc.

Testers: various testers included in the framework for different purposes. Before we examine these testers in detail, note that these testers belong to different strategies: Black Box, White Box and Grey Box testing. In the case of Black Box testing, the tester directly communicates with the application itself. But in the other two cases, additional information needed that must be provided by the programmer or directly extracted from the source code. In this situation, the tester communicates with the wrapper of application to obtain the required information (e.g. the application model). The wrapper and its usages will be explained in the following. Here is a non-exhaustive list of testers:

- **Functional Tester**: This is a Black Box tester that checks if the application behaves as expected. This tester simply applies test

cases to the application directly, and passes the generated results to the oracle to compare them with the expected behavior.

- **Code Coverage Tester**: this is a White Box tester that determines the adequacy of test cases by assessing the level of coverage of the structure they reach. Two common forms of code coverage are *function coverage*, which reports on functions executed and *statement coverage*, which reports on the number of lines executed to complete the test. Besides standard coverage measures, other coverage criteria can be defined for web-based applications that we discuss them in Model-Based tester. As this tester requires an instrumented version of application, it must communicate with the wrapper interface, asking for enabling the instrumented code (see Wrapper below). In some programming languages like java, the instrumentation can be done directly on the object code, hence there is no need to the source code (or the wrapper in our case).

- **Model-Based Tester**: a model describes some aspects of the system under test. The model is usually an abstract presentation of the application's designed behavior. Models have two usages in testing of web applications: automatic generation of test cases (see Test Case Generator below), and structural testing of application. In the latter, a high level representation of application, like the navigation model is used. Navigation model describes a web application using its composing pages (static and dynamic) and navigation links. Using this model, some coverage criteria such as page coverage and hyperlink coverage can be tested, in order to determine the adequacy of test cases. But how the tester acquires the model? The models can be built manually by the application programmer, or created automatically from the application (source) by running a special program. In each case, this is the responsibility of the wrapper to provide this model to the tester (see Wrapper below) via an appropriate interface, using STOL language.

- **Stress Tester**: this is a Black Box testing which checks for the stress the applications can withstand. The idea is to create an environment more demanding than the application would experience under normal work loads, and to see how application responds to this extreme load.

- **Load Tester**: this is a Black Box Testing in which the application is tested against heavy loads or inputs such as testing of web sites in order to find out at what point the application fails or at what point its performance degrades. Load testing operates at a predefined load level, usually the highest load that the system can accept while still functioning properly.

- **Security Tester**: security testing is carried out in order to find out how well the system can protect itself from unauthorized access, hacking, cracking, any code damage etc. In the case of web applications, the application must be tested under familiar attacks like SQL injection, session hijacking, XSS attacks and so on.

Test Case Generator: this component generates the inputs to the desired testers. In the case of a web application, test cases consist of URLs, name-value pairs (input data) and user actions (like clicking on a button). This component is a critical part for automating the test process, since we have anticipated four independent modules for this component in the framework.

1. The first module provides record and playback features that allow testers to record interactively user actions and replay it back any number of times (using Executer component), and comparing actual results to those expected (using Oracle component).
2. The second module exploits the user session logs of a web server as a means of generating test cases in a large scale. Each user request logged in the web

server usually consists of IP address, time stamp, the requested URL, cookies, name-value pairs in GET/POST requests and the referrer URL. By following the subsequent requests in a predetermined time interval from a particular IP address, a complete user session can be extracted and used as a new test case.

3. The third module uses different models and specifications of the application, like UML models, navigation model, Z-specifications and etc. to generate the test cases. The models or specifications can be obtained from the proper interface of Wrapper, using STOL.
4. And finally, the fourth module uses ontology-based test case generation approach, in which the ontology of the application domain is utilized to generate test data. For instance, in a web forum application, a light-weight ontology can be developed which describes the users, and then, different parts of the application, e.g. form input fields, can be annotated with concepts of this ontology. In such case, test-cases for those parts can be generated automatically based on the ontology.

Test Oracle: generates the desired outputs for the test cases and compares the actual results of tests, to the desired ones to determine the degree of success. In the case of web applications, the desired outputs mainly consist of HTTP responses returned from the web server. Generation of the desired outputs depends on the method selected for test case generation.

Executer: this component sends the test cases to the web server and receives the responses, and finally sends them to the Oracle to be evaluated.

Wrapper: An important problem in white-box or gray-box testing of an application is the lack of knowledge about internal structure of that application. Most programmers don't share their valuable source codes with other, and this is a problem for tests that require the source code. Because we want to present a framework to provide testing service for external application, we need to tackle with this problem. Our solution is simple: external applications should provide required wrappers to provide enough information for the framework to perform the test. Each time a tester requires a service that is beyond the functional services of the application (i.e. requires the source code), calls the appropriate API of the wrapper and receives that service. Some of theses services are:

- Activate the instrumented mode: some testers likes Code Coverage Tester require an instrumented version of application that includes instrumentation codes measure the coverage obtained during test. These testers have to activate the instrumented mode, before starting the test.
- Getting various application models: testers like Model-Based Tester require application models (e.g. navigation model) and they can obtain these models by calling this service with the name of their desired model. This service has an API that returns the list of models represented by this service.

To produce the wrapper, an application is distributed for each web programming platform (e.g. PHP, Java …). Developers can produce the wrapper by running this application on their source code, and add the resulting codes to their programs.

3. Conclusion and Future Works

In this paper a general framework for automatic testing of web-based application has been proposed. This framework contains different testing techniques, including black-box and white-box testing, which make it a comprehensive testing framework. Also we proposed a Wrapper for each application

that provides the required internal information to the testers. This Wrapper solves the problem of lack of trust between application developers and the external testers. Furthermore, we define a language and ontology called STOL (Software Testing Ontology Language) for communication between different parts of the framework.

For the future, we plan to (1) Define the details of different components of the framework and connections between them then, (2) implement and test the performance of the framework using current open source tools and programs written by team members.

Acknowledgement

This work has been supported by a grant by Iran's Telecommunication Research Center (ITRC), which is hereby acknowledged. The authors also appreciate the support of WTLab at FUM.

References

[1] Filippo Ricca, Paolo Tonella, "Web Testing: a Roadmap for the Empirical Research," *wse*, pp. 63-70, Seventh IEEE International Symposium on Web Site Evolution, 2005

[2] Sreedevi Sampath, Valentin Mihaylov, Amie Souter, Lori Pollock, "Composing a Framework to Automate Testing of Operational Web-Based Software," *icsm*, pp. 104-113, 20th IEEE International Conference on Software Maintenance (ICSM'04), 2004

[3] Hong Zhu, "A Framework for Service-Oriented Testing of Web Services," *compsac*, pp. 145-150, 30th Annual International Computer Software and Applications Conference (COMPSAC'06), 2006

[4] P. Dhavachelvan, G.V. Uma, "*Multi-agent-based integrated framework for intra-class Testing of object-oriented software*", Applied Soft Computing 5, pp. 205–222, 2005.

[5] Nuo Li, Qin-qin Ma, Ji Wu, Mao-zhong Jin, Chao Liu, "A Framework of Model-Driven Web Application Testing", Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06), 2006