Towards Call for Testing: An Application to User Acceptance Testing of Web Applications

Lian Yu¹, ¹Wei Zhao, Xiaofeng Di, Changzhu Kong, Wenbo Zhao, ²Qianxiang Wang, ¹Jun Zhu,
School of Software and Electronics, Peking University, Beijing, 102600, PRC
²Key Laboratory of High Confidence Software Technologies, Peking University
¹China Research Center, IBM, Beijing, 100094, PRC

Abstract

The paper proposes a new test model, Call-For-Testing (CFT), and applies it to User Acceptance Testing (UAT) of Web Applications. Different from traditional approaches, UAT in the context of CFT only means beta-testing on the site or field, i.e., by customers or end-users, and does not include alpha-testing which is supervised or performed by the system provider. CFT model leverages open community resources for UAT. To guarantee the quality of CFT-oriented UAT, not only do we develop competent testing techniques/tools for community testers to perform UAT, but also a testing auditing mechanism is built up. We develop a CFT platform that constructs coverage criteria based on user requirements and design documents, monitors community testers activities in real-time, and verifies their performances. Experiments are performed to evaluate the validity of CFT.

Keywords: User Acceptance Testing, Call for Testing, Test, Test Case Generation, Web Testing.

1. Introduction

The rapid growth of Web-based applications (WebApps) has revealed a lack of effective approach to performing the user acceptance testing (UAT). Especially, it is difficult to find proper end-users to participate in the UATs in terms of user volumes and classes, as WebApps are literally serving the global users.

This paper proposes a new testing model, Call-For-Testing (CFT), with the intentions to leverage the open community resources to perform UAT of WebApps. Open community, composed of a large amount of students, program development amateur and the likes, can be seen as virtual development resource pool. As a matter of fact, taking good use of these open community resources is becoming long-term goals of many businesses.

Different from traditional approaches, UAT in the context of CFT brings the following benefits:

- Everyone who uses Internet has an opportunity to participate in the UAT. Therefore, in theory, there is no limitation of user volumes and classes.
- CFT can achieve an ideal UTA, that is, to let the end-users create test cases and perform UTA without the supervising of WebApp developers.
- As the testers of CFT from open community can concurrently work with WebApp developers, the time-to-market can be cut down dramatically.

On the other hand, CFT leads to new issues and challenges to UAT of WebApps. In the environment of CFT, three main roles participate in the interactions: WebApp requirement owner, WebApp developers and community testers. To guarantee the quality of CFT-oriented UAT, not only should competent testing techniques/tools be developed for community testers to perform UAT, but also a collaboration platform among the three roles should be built up, and a testing auditing of CFT mechanism should be constructed. To support UAT in the context of CFT, this paper proposes the collaboration process among participants and provides them with suitable services as follows in Figure 1.

![Fig. 1. Collaboration process and of CFT platform](image)

1) Support the annotation activities by WebApp developers on Web browser: Allow developers to specify explicitly the types and validation points of Web UI elements; store the annotation results and send to the CFT server. In addition, WebApp developers can obtain requirements from the requirements owner,
design and implement the WebApp system. Once informed of bugs, they will fix them.

2) **Support Web UI testing by open community testers on Web browser**: Display the annotations provided by developers; record testing activities; send the test activity information back to the CFT server; report bugs to the bug tracking system, and remind testers of which UI element(s) is/are not tested yet according to the coverage analysis on the CFT server side.

3) **Perform test auditing on CFT server in terms of test coverage**: Obtain annotations provided by developers; parse HTML Web pages to pull out UI elements and relations annotated; and produce the test requirements. Parse test activities performed by testers and retrieve test results. Compare the tester’s testing activities with the test requirements, and assert the coverage in terms of UT element coverage, element relationship coverage, simple sequence/path coverage and complex sequence/path coverage. The coverage results can be reflected to testers as a reminder/warning.

The paper is organized as follows. Section 2 presents the architecture of CFT platform. Section 3 describes services for developers. Section 4 portrays services for community testers. Section 5 depicts services on CFT server. Section 6 illustrates the experiment results. Section 7 surveys the related work. Finally, Section 8 concludes the paper and outlines the future work.

### 2. Architecture of CFT Platform

Figure 2 shows the overall architecture of CFT platform, which consists of three parts: services provided for testers on browser side, services provided for developers on browser side, and services equipped on CFT server.

![Fig. 2. Architecture of CFT platform](image)

Based on the collaboration process presented in Figure 1, the following services are identified for each participant:

- **Services for developers**: Annotating service facilitates developers to specify a web page rendered on browser which elements need to be validated and in what way, which elements are associated with what else elements and in what way. Storing annotation service stocks up the annotation information, and uses Ajax to send back to CFT server.

- **Services for testers**: Displaying annotation/coverage service pulls annotations afforded by developers and displays on browser such that testers can generate the corresponding test cases; obtains CFT auditing results and highlights the not-yet-test elements on browser, and presents the rest of results in terms of coverage. Recording test activity service tracks tester’s motion on browser and uses sending test results service to store back to CFT server.

- **Services on CFT server**: Parsing HTML service crawls target under test (TUT), parses UI elements, such as buttons, links and events that are both explicitly written in HTML or dynamically added. Based on original requirements from the owner, UI design annotation provided by developers and UI elements and events parsed, generating test requirements service produces test requirements which the test auditing service checks against, and computes test coverage.

The following three sections will describe each of three parts in details.

### 3. Services on Developer Side

The services on developer side allow developers to annotate requirements or design information on a Web page. The annotations are stored into CFT server and retrieved by testers later, such that testers can look up them as their focus of testing tasks; on the other hand, the server can use them to set up test requirements as to audit the testing quality of the testers.

Services on developer side provide two functions as shown in Figure 3: 1) Supply pop-up Web forms to enable developer to write annotations. Each type of information is annotated via a form format designed specifically. As in Figure 3, a controller object selects a proper form to pop up according to developer’s selection. 2) Communicate annotations with CFT server.

![Fig. 3. Modules of services on developer side](image)

#### 3.1. Annotating Service

Annotating service shores up Web developers to label three types of page design information: text input elements, dynamic script events and relationship of a group of page elements:

1) **Text input elements**: The well-used way for Web users to input information is to use the text input
element on Web pages. User’s inputs may be characters. In an application, however, the characters may be regarded as different data types, such as integer, decimal, date or string. Moreover, there may be more stringent constraints on these actual data types. For example, a number may have a maximum or minimum limit; a date is required within a range; a string needs to conform to a pattern. The constraints related to data types based on requirements need to be annotated by Web developers.

2) Dynamic script events. In Web 2.0 days, large numbers of script events are added into Web pages to improve user interaction effect. Those events are listed in the HTML specification. They are triggered by user’s action on pages. The annotation about a page element associated with an event would remind a tester to test it.

3) Relation of a group of page elements. A group of page elements may be designed to work together to serve some purposes. This paper identifies four types of relationships as follows.
   - **Inclusion Relation.** If one element includes two or more elements logically, the element and the others included are in an inclusion relation.
   - **Dependency Relation.** If the state or attribute value of one or more elements depend on the state or attribute value of other element’s, these dependent elements and the depended element are in a dependency relation. For example, if “China” is selected in a pull-down menu, cities in China are listed in a sub-pull-down menu for further selection.
   - **Coexistence Relation.** If two elements exist or do not exist at the same time, the two elements are in a coexistence relation. In other words, if one element exists, the other one must exist; if one element does not exist, neither does the other one definitely. Existing or not is represented by a state, such as in an enabling or active state.
   - **Exclusion Relation.** If two elements are always in opposite states, the two elements are in an exclusion relation.

### 3.2. Communicating Service

Annotation forms are implemented as HTML form elements. When submitting a form, annotation information is sent to CFT server by HTTP post method. If a type of annotation information of an element has been stored on the server side, before the form pops up, the annotation information is obtained from CFT server by Ajax asynchronous method which is wrapped in an AjaxObject. Web developers can modify the old annotation and save it again. If there is nothing about the element on the server side, an empty form is shown to a developer and he can create a new annotation about the element. Services on developer side are implemented in JavaScript language. The script needs to be loaded from CFT server first and then the developer can use the services.

### 4. Services on Tester Side

Services on tester side record the tester’s actions on Web pages and send the testing information to CFT server. On the other hand, it is similar to the communication service on developer side that the Displaying Service shows to testers the annotation related to an annotation type and testing coverage results.

Note that the reporting bug service is to supply the functions of bug tracking, not the focal point of this paper.

### 4.1. Displaying Annotation/Coverage Service

Displaying annotation service obtains annotation information contributed by Web developers and stored on CFT server. Testers can know the key testing points based on the annotation about requirements and design information; however, testers can not modify the annotation as developers do. On the other hand, displaying coverage service allows testers to view the testing coverage that is calculated by testing auditing service on CFT server. The online auditing facilitates testers to decide what-to-do to improve their test coverage.

### 4.2. Recording Test Activity Service

A variety of event handlers are developed to capture tester’s actions, e.g., clicking a link element, or inputing data, such as text input data and radio box. Different event handlers are developed to handle different kinds of test object elements. The test object elements are added to proper event handlers according the tester’s action or input data. The handlers are connected to elements once the tester side script is loaded.

### 4.3. Sending Test Result Service

For each kind of test object elements, there is a TestResultEntity object holding the testing data. Once a tester completes the action on an element, an AjaxObject sends the testing data in JSON format (http://json.org/) to CFT server.

### 5. Services on CFT Server

In order to perform test auditing, CFT needs to prepare test requirements and check them against testing data received from client script. The task is broken down to three services: 1) Parse Web pages to extract UI elements; 2) Generate test requirements based UI elements extracted, annotations by developer and navigation diagram from requirements; 3) Audit testing activities by testers against the test requirements in terms of test coverage.

### 5.1. Parsing HTML Service

The service crawls all Web pages of WebApp under test, and extracts the elements of links, inputs and events.
5.2. Generating Test Requirement Service

The section describes generating test requirements for link, event, input, element relation, simple path and complex path.

Generating Link, Event, Input and Relation

Link information is parsed from HTML code. The same URL may be depicted in different text or images. So test requirement for link is represented in <url, text> format.

Event information is from HTML code and annotations. If the event is connected to a page element as an attribute of the element, parsing HTML service can parse it. To get the events dynamically added, we embed Rhino (http://www.mozilla.org/rhino/), a JavaScript engine from Mozilla, in the web page analysis module to parse and execute JavaScript code on a web page. Then we can get both statically and dynamically added events from Rhino. It is also annotated by developers. So test requirements for event are generated from the two sources, in the representation format of <element tag name, id attribute value, name attribute value, event name>.

Input elements are divided into three categories: text input elements, single choice elements and multiple choice elements.

1) Text Input Elements: They include the input elements such as text, password and textarea elements. The constraints of the elements are annotated by developers. We use equivalence partition method to generate its test requirements.

2) Single choice elements: They include radio button and select list with single choice. Usually, the value of every choice item is written in HTML document. Test requirements for this category are the choice values.

3) Multiple choice elements: They include checkbox and select list with multiple choices. Multiple choice means combination among choice items.

Relation is annotated to indicate relationship among several elements. Test requirements for relation describe element’s roles in the relation. The representation is <relation type, relation id, (role, element tag name, id attribute value, element value)>.

Generating Simple Path and Complex Path

CFT server uses a navigation diagram representing use cases of requirements as input to generate simple paths and complex paths. Figure 4 shows an example of such a navigation diagram, where the circles represent Web pages and arrows represent navigation conditions, such as clicking links. We distinguish paths into three levels. The first level is simple path, not containing loops. In Figure 4, simple paths include ab, ac, and acd. The second level is simple loop, which can be represented in regular expression. In Figure 4, simple loops are (ac)+ and (acd)+. In generating simple path, the path is stopped extending once a loop is found or a leaf node is reached.

The third level is called complex path, equivalent to sub-graphs, which is formed by combining a simple loop with another paths. The simple loop and another path have a common node at least. Complex path can be represented in regular expressions, which is a conversion problem between finite state machine and regular expression [1].

Fig. 4. Navigation diagram for use case example

The idea is to remove non-leaf-node in the graph constantly, and the edge ending with the removed node and the edge starting from the removed node are merged together. So finally root and leaf nodes are left, and the expression of the edge from root to a leaf node is the regular expression of a complex path. The basic merging patterns are shown in the Figure 5.

Fig. 5. Basic merging patterns

5.3. Test Auditing Service

Testing results data from testers is to be compared with test requirements generated by CFT server to audit whether one of the test requirements is tested adequately. If a group of testing data is determined to accord with one of the test requirements, set the corresponding field of the test requirement in database as 1. The flag field is used to compute the test coverage.

6. Experiments

To evaluate CFT platform, this section presents experiments to demonstrate its functionality, testing auditing ability and event analysis capability.
6.1. Auditing Test Coverage

To demonstrate the auditing capability of CFT platform, we perform an experiment with three testers, A, B and C. A developer annotates Amazon homepage and stores into CFT server. Textual descriptions of requirements with three use cases: user login, product purchase, and look-up help are given to the three testers. The developer annotates one input, 21 events and 22 relations. CFT server produces test requirements for Amazon homepage in terms of links, events, inputs and relations, in addition to simple paths and complex paths based on a navigation diagram as shown in Table 1.

Table 1: Number of test requirements produced by CFT

<table>
<thead>
<tr>
<th>Number</th>
<th>Link</th>
<th>Event</th>
<th>Relation</th>
<th>Input</th>
<th>Single path</th>
<th>Complex path</th>
</tr>
</thead>
<tbody>
<tr>
<td>283</td>
<td>21</td>
<td>22</td>
<td>32</td>
<td>12</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Three testers are required to test Amazon homepage annotated by a developer regarding to link, event, input and relation, and test the three use cases.

![Fig. 6. Test coverage for the three testers](image1)

It is observed from Figure 6 that Tester C accomplishes the best with 70% coverage on average, while Tester A only obtains about 35% coverage on average. Three of them reach 70% coverage of link, modest level coverage for event and relation. The coverage of complex path gets the lowest score for three of them, which indicates the coverage of complex path is difficult to realize and cost much.

To analyze the relation between test coverage and fault detection capability, we track within 4 successive days a tester work on testing an e-commerce Website, called PetStore, where 8 faults are seeded with different types as shown in the sixth row of Table 1. Figure 7 records six types of coverage along the 4 days.

![Fig. 7. Test coverage along 4 days for a tester](image2)

It is observed due to the complete annotation of statically added events and dynamically added events by the developers, the tester achieves a good coverage on events up till 100%; compared with complex path which gets lowest coverage during the first three days, simple path is easier to attain 100% coverage. Input coverage is even lower complex path at last, as input has a wider array of variation, such as types, formats, and ranges. Link coverage and relation coverage accomplish modest levels.

Table 2 demonstrates the number of faults detected along the 4 days. Event and simple path achieve 100% coverage and 100% fault detection. Input achieves lowest coverage but acquires 100% fault detection. Faults associated with relation and complex path can not be detected.

Table 2: Number of faults detected within the 4 days

<table>
<thead>
<tr>
<th>Number</th>
<th>Link</th>
<th>Event</th>
<th>Relation</th>
<th>Input</th>
<th>Single path</th>
<th>Complex path</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

6.2. Analyzing Dynamically Added Events

An event can be attached to an element on a web page by setting attributes or by adding event listeners. Events attached by the latter method is usually called dynamically added events, events attached by the former method is called statically added events.

To verify the capability of detecting dynamic events on Web pages, we carry out the following experiment. First, we select eight websites which are commonly used in the world, including Ebay (http://www.ebay.com/), Amazon (http://www.amazon.com/), Apple (http://www.apple.com/), myspace (http://www.myspace.com), Mozilla (http://www.mozilla.com/), Digg (http://digg.com/), Alibaba (http://www.alibaba.com/), and Yahoo (http://www.yahoo.com/).

Then we check the attributes of all the elements on each Web page using Web analysis module without considering dynamically added events. So we can know how many events are attached statically by counting the number of events that the analysis module detects. After that, we use the Web page analysis module with the dynamically added events to count the number of events both statically and dynamically added on each of the eight homepages, and compute the sum of statically added events and the sum of all events for the eight Web homepages as shown in Figure 8.

It is observed that Amazon, Apple and MySpace have more than 90% of statically added events on homepages, and Digg reaches 41%. On the other hand, Ebay, Mozilla, Alibaba and Yahoo only have less than 1% of statically added events on homepages, and the rest are dynamically added events. Without the capability of detecting dynamic events, we would lose most of test requirements for events. It is also interesting to notice that the homepages heavily
using dynamically added events tend to have more events on the pages, for example, Mozilla, Alibaba and Yahoo have more events on their homepages than Amazon, Apple and MySpace do.

![Fig. 8. Analyzing dynamically added events](image)

7. Related Works

UTest (http://www.utest.com) has the similar idea with ours. Leverage community resources, company can build virtual testing team on demand, define testing requirements and get real-time valuable bug reporting and application feedback from the uTester global community. UTest supplies complete testing process management, including performance testing and load testing; CFT is expected to be a collaboration platform, focusing on user acceptance testing, especially on Web UI testing now. CFT supports simple and convenient annotation mechanism for developers to describe requirements, avoiding traditional lengthy and detail requirement documents. UTest has maturity indicator including many parameters to grade the maturity of application under test. CFT uses test coverage to audit community tester’s activities and reflects the degree of application under test is exercised. Indeed, coverage only does not fully indicate testing adequacy. The number and quality of bug detected are also import indicators. We will integrate bug tracking system into CFT in the future.

Coverage is introduced in white-box testing first, including control-flow coverage [2] and data-flow coverage [3]. Later coverage is expanded into black-box testing gradually. In UML-based testing techniques, coverage is applied in many UML diagrams, such as class diagram [4], sequence diagram [5], collaboration diagram [4], activity diagram [6] and state diagram [7]. Paper [8] defines a group of GUI event-based coverage criteria. Page flow model is represented in paper [9], in which a group of coverage criteria to Web application are based. In CFT, we define coverage related to all kinds of test targets, including link, event, input value, relation and path.

8. Conclusion and Future Work

The paper proposes a new testing model, call-for-testing (CFT), and applies it to Web user acceptance testing. CFT leverages the open community resource to contribute to Web testing, and imposes a lot of challenges to implement the model. CFT needs not only distributed testing infrastructure but also requires a mechanism to assess the testing quality of testers from open community. This paper develops a CFT platform, which allows Web developers to annotate Web pages in terms of inputs, events and relations of UI elements. Testers can use the annotated information together with requirements and perform Web testing. The tester’s testing results are recorded and stored in CFT server. Based on a variety of coverage defined, CFT server generates test requirements based on requirements and the annotated information, and audits testing coverage for each tester. We perform experiments to evaluate CFT functionality, auditing capability and analytical ability to capture dynamically added events. The future work is to integrate with bug tracking systems.

References