Automated Functional Testing of Search Engine

Lingzi Jin
New Media Technology Department, Yell Group, Queens Walk, Reading, UK, RG1 7PT

Abstract

This paper describes the construction of an automated test framework for search engine and reports our current practice in establishing a process for test automation. The paper presents the technical solutions that overcome the difficulties in search engine testing, which involve large volume of data, complicated ranking rules and randomization in ordering search results. One reason that many test automation efforts failed in industry is because automated scripts are not developed and structured for future maintenance and evolution. The test framework enables effective and efficient reuse and maintenance of test scripts. The paper also discusses issues related to the process of test automation. They are equally as important as overcoming technical barriers. All of these discussed above have been applied to the automated functional testing of a commercial search engine.

1. Background

A web search engine is a tool designed to search for information on the World Wide Web. There are different ways that search engines work, but they all perform the following tasks [1]:

(a) Crawl the Web and download web pages from the Internet.
(b) Extract the keywords from downloaded pages and build an index of the keywords and the address of the page.
(c) Allow users to query for information using keywords or combinations of keywords in that index.
(d) Display search results according to ranking rules, which is often randomized to achieve fair listing of equally best matches.

A commercial search engine may index hundreds of millions of pages, and respond to tens of millions of search queries per day. Good quality assurance is the key to ensure it will be functioning. Among other things, efficiency and effectiveness are the primary concerns for testing. Test automation offers a technical solution to these concerns [2, 3].

This paper discusses the challenges of test automation for search engines and reports our experiences in development and application of test automation principles and techniques to improve the test effectiveness and efficiency as well as the management aspect in test automation projects.

2. Challenges in search engine testing

Functional testing of a search engine is difficult, especially in defining the expected result due to the following reasons:

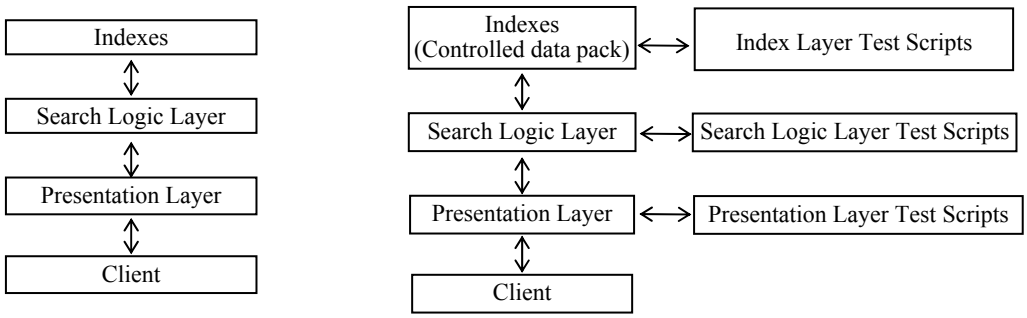
- Massive amount of data are stored in indexes which come from various sources.
- Complex search logic is applied to filter out the search result from data against search terms.
- Ranking rules are used to control the order in which results are displayed. They take into consideration of relevance and product types. For example, a sponsored listing is always displayed on a top position for relevant keywords. However, they do not uniquely determine the display order. In fact, randomization often takes place.

Our solution to this challenge is to establish a "*controlled data pack*" on which search engine operates in test executions. This means to populate a database for testing purposes with specified data that fairly represent the real database, but easier to manage in testing process and to define the expected behaviors.

Like all web-based applications, the search engine, especially its presentation layer, is under continuous evolution and frequent changes. Scripts for automated testing often have to be modified every time the software under test is modified. Therefore, how to manage test scripts to enable their evolution and reuse imposes the second challenge. Our solution is to adopt a layered architecture of test scripts; see Section 3 for details.

We regard test scripts as an important software artifact and valuable asset. Test scripts will only be thrown away when the software under test is out of use. For large scale software systems, test script development is costly and labor intensive. Our experience shows that their development can be a complicated process and must also be managed with proper quality assurance that is integrated into the whole software development/QA process. This imposes the third challenge. This issue is addressed in our process model of test automation and discussed in Section 4.

AST'09, May 18–19, 2009, Vancouver, Canada
978-1-4244-3711-5/09/$25.00 © 2009 IEEE



(a) Search engine (b) Test structure

Figure 1. Architecture of search engine and test levels

3. The architecture of test automation

3.1 Overall strategy

Logically speaking, a search engine can be depicted as a 3-tier architecture in Figure 1(a) [4, 5]. According to this architecture, our strategy of testing search engines can be summarized as follows.

First, indexes from *Controlled Data Pack* replace the real indexes in the automation test environment. It is obtained by extracting a subset of the real data. The extraction process consists of two steps. The first step is to search the real indexes on the test cases. The second step is to manually examine the results of the first step and select a subset of the results so the data cover various typical situations and boundary conditions. Therefore, the data in the Controlled Data Pack are representative and of adequate coverage. A set of indexes are generated from the Controlled Data pack and used in testing.

Index queries are API calls. Their functional correctness is tested by sending queries to the index server to invoke the APIs and then examining the returned xpath values [6] in XML format. Such test scripts corresponds to the Indexes Layer Test Scripts in Figure 1(b).

For the search logic tier, test scripts were written to send queries to the web server and check the return value of xpath values in XML. These test scripts form the Search Logic Layer tests depicted in Figure 1(b).

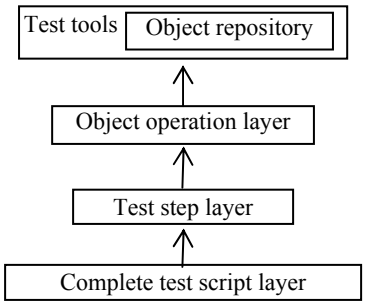


Figure 2. A layered architecture of test scripts for presentation tier

Finally, for the presentation tier, since it is what end user will see and use, it obviously becomes the part where most testing efforts are spent. However, the front end testing is the trickiest task because it is subject to frequent changes. There are two requirements of its test scripts.

A. Robustness: All automated tests should be run unattended overnight as one batch from beginning to end without abnormal termination even when critical defects exist in software under test. In particular, scripts should be able to handle things such as waiting for web objects to be loaded and if an expected control doesn't appear, the batch run can still carry on after the problem is logged.

B. Maintainability: Software feature could change between releases, so automated test scripts must evolve accordingly. Scripts should be easy to maintain when update is required.

These requirements for presentation layer testing are met by the layered architecture of test scripts depicted in Figure 2. The following subsection explains how the architecture works.

3.2 Object operations

To achieve robustness and maintainability, our test framework takes advantage of object orientation. In particular, polymorphic operations can be applied to all objects in the form of:

```
Object. Click
Object. Set Value
Object. Select Index
    .....
```

However, such operations are not robust enough. For reliable executions of scripts, every object needs to check its availability before an operation is applied and bug information must be recorded when the software under test fails. Code was developed to provide the robust basic operations on objects. The following is an example of ClickObject function:

```
Function ClickObject (Object, ErrInfo)
    With Object do
        If .WaitProperty("disable", 0, 15000) = False
```

```
        Then
            Reporter.ReportEvent (1, " object  not displayed" ,
                ErrInfo)
            ClickObject =1
        Else
            .Click
            ClickObject =0
        End If
    End with
End Function
```

In this layer, more specific functions were written for particular objects, such as ClickWebLink and ClickWebButton, which all call the ClickObject function.

3.3 Test steps

A test step represents a set of actions that form a state transition in the operation of the system. It is normally in the following format:

```
Pre-condition, e.g. a certain window is displayed
Setup Actions, e.g. enter data into editing fields, and/or select
data from drop down lists or radio buttons.
Transition Action: e.g. click a submit button
```

The setting up of all data in one step should be encapsulated into one function so that the impact of any future GUI changes such as adding/removing a field will be localised and changes are kept to the minimum. The return value from the function will indicate whether those operations have been performed as expected such as reporting to the caller that a mandatory field is missing.

These functions are implemented on top of the object operations layer.

3.4 Test scripts

A test script is a test case which consists of a sequence of steps of state transitions. It is implemented by a sequence of function calls and check points. Depending on failure or pass of the check points, it will decide whether it will carry on to do next test step or stop current script run. If the current test cannot continue, such as expected window didn't get displayed or a crash, the script should be able to clean up states and launch web browser again to start next test. This layer also includes the common checkpoint functions.

It is worth noting that, two sets of automated test scripts are developed. One set checks the full details of every result listing. They can only be run against the specific test automation website with controlled data pack indexes as described in section 4.2. The other only does a limited checking, but can be run against all manual test websites, the soft live website (internal beta) and the live website which all have real data. The second set, called generic pack, proves to be the most popular and get most credit for test automation, since it is useful for all people including the support team outside the development/QA team. It also provides a safety net for confidence in quality in all environments.

3.5 Object repository

Some commercial test automation tools such as HP Quick Test Professional (QTP) [7] require users to manage object repositories. An object repository shows all objects used by the software under test and their test related information, including how the test automation tool recognises them and which properties can be used in scripts. When GUI changes come up, both scripts and the object repository may have to be updated. Otherwise, QTP could complain that the new object referred in the script is not in the repository and script run will fail.

A centralised (shared) object repository will help since any repository change caused by GUI changes will be made just in one place.

4. Process issues

Many organizations have tried to implement test automation. Some failed but others not. Some bought test automation tools but left on the shelf forever after tried it for a few months. There are technical reasons as well as process issues.

4.1 Test asset evolution process

Automated testing takes time and effort to develop and maintain. From our own experience, an automated script takes about 5-10 times to develop than a manual one. Given the fact that the benefits of automated testing may not be widely recognised, we have taken an incremental approach to gradually build up a set of tests with good coverage and yet at the same time, use them and benefit.

The initial automated test set starts from identifying those features which are critical to business functions and required to run for every new release. Then, a feedback loop is established to grow and assess the effectiveness of automated tests. Since feedbacks are from various sources, the automated test set evolves towards the right direction and demonstrates its value.

In current industrial environment, the development of the web-based software consists of a number of projects which follow either Waterfall or Agile development processes. Testing requirements and feedbacks come from the following sources:

- QA leads of Waterfall project or Agile scrum sprint;
- Customer support team for live site issues.

After a new project or a sprint finishes, the QA leads provide a regression test pack to the test automation team. The test cases are then reviewed. Automated test cases are identified according to the following principles.

A. Long term value. We only automate those tests that have a long term value and will be repeatedly used in the future releases. Here long term means those features will

stay in our web site for next 6 months at least. Short term features do not worth automated tests.

B. Less Duplication. Test cases and test scripts often need refactoring and reorganisation. For example, we organise test cases according to the underlying code sequences. Unnecessary repetitive tests and steps are removed. This not only improves the test effectiveness, but also reduces maintenance cost because it will keep a compact set of tests.

When a new release is done, we need to review existing automated tests. In particular, obsolete features are identified. Test priorities are reassessed and adjusted accordingly. Some scripts could be removed and updated as a result of this process.

4.2 Management of test assets

Automated tests often run in a dedicated environment. Our test automation environment consists of a test website as described in 3.1, and multiple client machines with different browsers installed so that automated scripts can be run on all supported browsers. Attention has been paid to make this environment as consistent as possible to other manual test environments and the live website all the time. Any discrepancy with live website should be aware of.

This test automation website has its own indexes generated from the controlled data pack. To keep it "controlled", a data map is maintained, which contains all details of the data so that they can be viewed at different levels of abstraction and maximally reused.

It is inevitable that automated scripts need to be maintained to keep them up-to-date. The automated testing team is often under pressure to update test scripts in time to meet release deadlines. The architecture described in the previous sections makes this task a bit easier.

Another test asset that requires management efforts is the run time reports, or logs. Test automation can generate a large number of logs. After executing test scripts for a build, every failure reported in the log needs to be carefully investigated to identify its root cause.

5. Conclusion and further work

We have set up a test automation system and successfully run it for over a year. The automated GUI scripts are done in HP QuickTest Professional (QTP) [7]. Our experience demonstrated that automated functional testing has the following advantages:

(a) Faster test executions,
(b) Higher test accuracy,
(c) Better test accountability,
(d) Rigorously codified product knowledge.

There are also pitfalls that automated testing must carefully avoid. First, automated testing could be costly. This includes tool licenses, test environments and human resources. Moreover, developing and maintaining test scripts and managing test environment are time consuming. Our experience shows that a test script is cost effective only when it is executed at least 10 times.

Secondly, automated testing could be ineffective since they only cover what have been scripted. To achieve test effectiveness, automated test cases must be carefully designed so that we can find as many major defects as possible in limited number of test runs.

We are exploring other alternative test tools such as open source software Selenium Remote Control (RC) and JUnit [8] as our ongoing effort to reduce costs.

References

[1] Levene, M., *An Introduction to Search Engines and Web Navigation*. Pearson, 2005.

[2] Dustin, E., Rashka, J., Paul, J., *Automated Software Testing: Introduction, Management, and Performance*, Addison-Wesley, 1999.

[3] Fewster, M, Graham, D., *Software Test Automation*, ACM Press, 1999.

[4] Shklar, L., Rosen, R., *Web Application Architecture: Principles, Protocols and Practices*, Wiley, 2003.

[5] Kappel, G., Prýýll, B., Reich, S., Retschitzegger, W. (eds.), *Web Engineering: The Discipline of Systematic Development of Web Applications*, Wiley, 2006.

[6] W3C, *XML Path Language* (*XPath*) 2.0, January 2007, Available online at http://www.w3.org/TR/xpath20/.

[7] Mercury, *Optimize: Mercury Quicktest Professional: User's Guide*, Mercury Interactive Corporation, 2007.

[8] Hamill, P., *Unit Test Frameworks*, O'Reilly Media, 2004.