The IEEE International Conference on
Industrial Informatics (INDIN 2008)
DCC, Daejeon, Korea July 13-16, 2008

A Test Automation Solution on GUI Functional Test

[1]ZHU Xiaochun, [1]ZHOU Bo, [1]LI Juefeng, [2]GAO Qiu
[1]College of Computer Science, Zhejiang University, Hangzhou, P.R. China
[2]State Street Technology Zhejiang, Hangzhou, P.R. China
[1]{xzhu, bzhou, lijuefeng}@zju.edu.cn
[2]qgao@statestreet.com

Abstract- **Although it is widely believed that software quality will be improved by the use of automated testing, automation is still not well-off in industry today. There are quite a few issues of traditional software test automation mode, for example, high cost, knowledge barriers, and management troubles.**

In the paper, a test automation solution on GUI functional test is proposed. The solution integrates test case generation and selection, test case execution, and test reporting to facilitate testing. It introduces the concept of test driver which is designed to take over the communication between test cases and the execution engine. At last, some enhancements are proposed for future work.

I. INTRODUCTION

Although it is widely believed that software quality will be improved by the use of automated testing, automation is still not well-off in industry today. Currently, software test automation is managed and implemented by tasks individually in industry. When an automation task is raised, test engineers start to analyze the requirements, generate or select test cases, choose a tool to develop scripts, manage to run tests usually by mapping test scripts with test cases, and maintain scripts when changes come. There are several shortages of this mode:

Firstly, it is expensive to go through the whole procedure of designing, developing, and executing test scripts for every task, although sometimes it doesn't cost much just to finish one or two tasks. Especially, maintenance will be really expensive when big changes come. Additionally, low reuse and inflexibility of fixed scripts lead to high cost which prevents managers to choose automation. As shown in a survey of Australia in 2004 [1], about 50% of the respondents reported that cost was the major barrier to use automation for software testing.

Secondly, it sets the barrier to involve testers who don't have related programming skills. Specific programming skills such as a tool or a language are required to development scripts. So test automation becomes a game only for the group of automation developers. It requires that members must know test requirements well besides programming skills. They take response for both understanding test requirements and developing automation scripts accordingly. There is a risk that automation developers might misunderstand test requirements just like application developers often misunderstand business requirements.

Thirdly, given scripts and test cases are two separate deliverables, it is not easy to manage automated tests consistently. It is always a tough job to figure test cases to be automated and map them with corresponding test scripts based on different test strategies in each test cycle.

To overcome above disadvantages, different solutions are proposed by researchers and industrial companies.

Several kinds of test automation frameworks [2] are proposed to increase reuse and flexibility so as to reduce cost. Data-driven framework and Key-word/table driven framework are the most popular two. Sometimes the two are combined together to meet both data-based test and action-based test.

Dozens of techniques are created to support test case design and generation. For test case generation, the way of virtualized presentation [3] with formalized models tends to be a hot one. The State Charts which extend state-transition diagrams are used to model software specification [4]. Control Natural Language (CNL) is proposed to specify test requirements and test cases [5]. Test case design and generation solutions can involve testers even business people into test automation at an early stage so as to reduce test requirement analysis efforts of automation developers.

Various industrial tools are developed to support automatic test execution such as Rational Functional Tester from IBM and Quick Test Professional from Mercury [6]. The tools can run scripts which developed by them or written by languages they support. Besides, IBM and Mercury provide their own products to support test management: Rational Test Manager and Quality Center. The basic idea of them is quite similar that mapping test cases with test scripts and then configuring the scope of execution. Once a test case changes, manual effort should be introduced to update corresponding test scripts.

In the paper, a new test automation solution on GUI functional test is proposed. The idea is to put above ideas including test cases generation and selection, test execution, and test reporting together. The key is how to make the pieces automatically integrated. We introduce a test driver which is designed to interpret test cases into class objects and route them into a test engine which will then execute them automatically. How a test driver works will be presented in detail.

The reminder of the paper is organized as below. In section 2, background of GUI test is introduced. In section 3, our test automation solution is presented in detail including test case management dashboard, test driver, test execution engine and report engine. At last, discussion and future work are given in section 4.

II. Background

GUI functional test means validating GUI objects, checking functional flows by operating GUI objects, and verifying output data which are generated in backend and then displayed in front pages.

Here, GUI objects mean the elements in GUI clients such as a text, a dropdown list, a button and so on.

In manual testing, GUI testing is straightforward and relatively easy since all the actions are visible. But sometimes, same scenarios with large volume test data sets make trouble to testers. Too many repeatable tests are challenge to man. It doesn't matter for machine to do so. That's one reason that we'd like to automate GUI testing. Another reason is for regression test purpose. Once we set up automation scripts, we can run them just by clicking a button without any dedicated human resource in regression test.

The idea of software test automation is to let computer simulate what human do when manually running a test on the target application. A challenge of GUI test automation is how to recognize the GUI objects and the actions of them by machine. Fortunately, quite a few tools such as Rational Functional Tester (RFT) provided by IBM and Quick Test Professional (QTP) provided by Mercury are powerful enough to do so. All what we need is to make good use of them.

III. A Test Automation Solution on GUI Functional Test

A. Overview

Figure 1 gives a big picture of our functional test automation solution on GUI test.

There are different layers in the solution. At the top is test case management dashboard. It is used for test case design and test scope configuration. Once test cases are identified to be automated, a "device" named Test Driver starts its work to translate test cases into executable scripts such as Java codes in our practice. Then Test Driver routes the executable scripts into an execution engine for running. The execution engine receives scripts from the driver and automatically executes the tests. Here we use Rational Functional Tester (RFT) as our execution engine and codes are transmitted into RFT through its application program interface. Logs and results are recorded in the running time, and the data are used to generate reports.

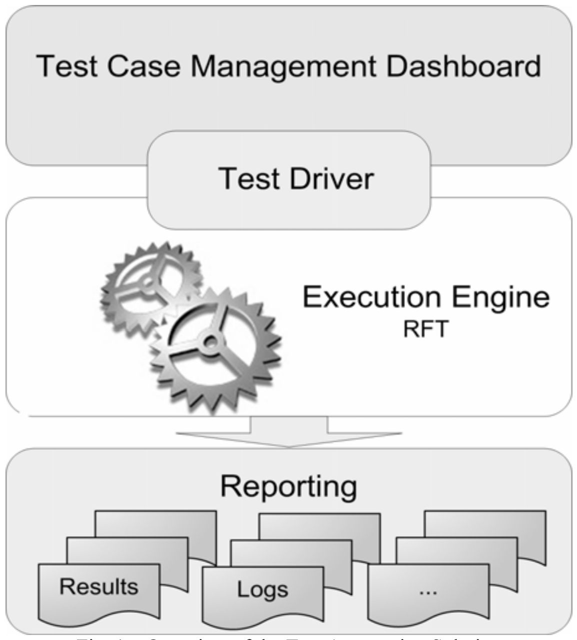

Fig. 1. Overview of the Test Automation Solution

From above, we have the basic idea about our solution which not only focus on test execution but also provides test management services. What's more, it doesn't require test scripts development phase because executable codes are automatically generated. In following sections, we will introduce each part in detail.

B. Test Case Management Dashboard

Test Case Management Dashboard deals with test case design, test case organization and test scope configuration. In our solution, we develop a web-based UI for the dashboard.

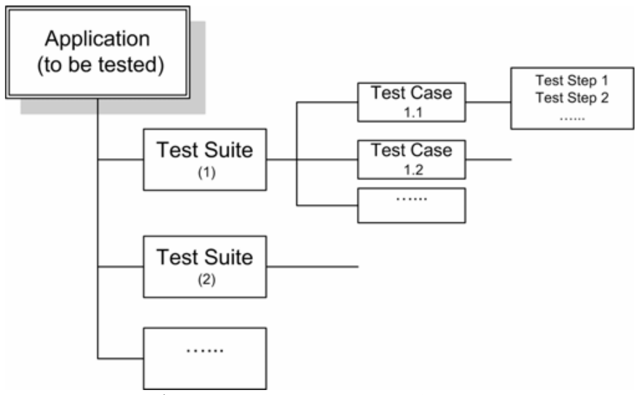

Fig. 2 Test Case Management Structure

Test Case Management

About test case organization, we use a top-down structure as shown in Figure 2. Applications to be tested are in the first level. In each application, different test suites will be created corresponding to each function component or Use Case. A Test Suite contains a set of test cases which are generated to test desired scenarios. For each test case, it contains several test steps together with test data to describe a test scenario. By the organization structure we can easily get the information of total case number and how many cases for each piece of functionality.

To identify automation scope, we make it configurable in test case level. It means we can select target test cases from each test suite via the dashboard. Once we decide how much test automation is enough, we can configure them accordingly. Of course, it is easy to generate the report that how many cases of each test suite are to be executed.

Test Case Design

In manual test, we describe test cases using natural languages as human can understand the meaning. But it is nearly impossible for any program or any tool to recognize our descriptions by natural languages. In order to let our Test Driver interpret our test cases well for RFT, we should use some formal method. In another side, we can't use too complicated languages for test case descriptions, otherwise it make no sense. In our solution we use the key-word driven framework to support this.

Sometimes, software engineers will use Controlled Natural Language (CNL) [5] to specify requirements or test cases. In a simplified way, each sentence (test step) in the specification conforms to the following structure: a main verb and zero or more arguments. A key-word description is just one kind of CNL. Usually, we define the Verb as one Key-word, and define elements after the verb as arguments.

In GUI testing, all actions are to operate GUI objects. In our solution, we define three basic key-words in a high abstract level as shown in table 1.

TABLE I
KEY-WORDS IN GUI FUNCTIONAL TEST AUTOMATION SLUTION

ID	Key-Word	Description
1	SET	Set values to each GUI object, such as input texts to an edit box, select an item from a dropdown list, click a button, uncheck a check box, etc
2	VERIFY	Verify system responses with expected results, such as data, object behaviors and so on.
3	INTERFACE	Provide mechanisms to be compatible of existing test scripts or to allow simulators for some test scenarios

Besides key-word, we have three parameters for the CNL: Object, Value, and Description. Object means GUI objects. Value, properties of an object, which in the meanwhile stands for test dada, can be a single value, a range of data, or even a data file. Description is not necessary during executing but helpful for comprehension. Each step means an operation to a property of a GUI object. So we can present a test step as a 4-dimension vector: <Key-word, Object, Value, Description>.

Usually, we present key-word test steps via a table. That's why sometimes, key-word driven is called key-word table driven too. For example, in case to describe a scenario that if we input "A" to EditBox1 and then select "B" from DropdownList1, Text1 will display "C", we can present it as table 2.

TABLE II
A SAMPLE OF TEST CASE DESCRIPTION BY KEY-WORDS

Step	Key-word	Object	Value	Description
1	SET	EditBox1	A	Input A
2	SET	DropdownList1	B	Select B
3	VERIFY	Text1	C	Verify output

In this way, testers who don't have programming skills can still design test cases to drive test automation. Even business analysts or users can join into test automation design with simple instructions.

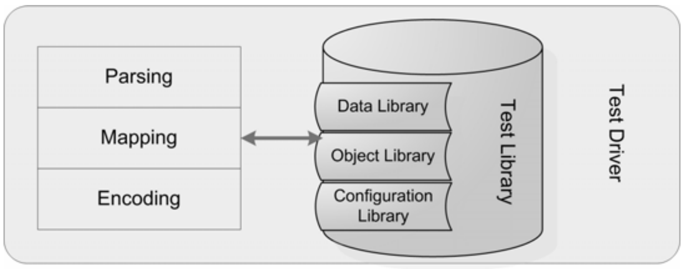

Fig. 3 Test Driver

C. *Test Driver*

When test case and test data are ready, it is time to execute them automatically. A question is how to make them recognized by automation engine. Here test driver helps to do the job.

Test Driver is the core of our approach. From figure 3, we see that in our solution, test driver is composed of test library part and driver function part.

Test library includes Data Library which contains test elements, Object Library which contains GUI objects information, and Configuration Library which contains application URL, application server and database information.

Object library is the key one in the driver. As discussed before, a key step to automate GUI test is to let machine recognize GUI objects. And Rational Functional Tester can recognize GUI objects when it touches and records them before. It is the same case for Quick Test Professional. That's why we should create object library of GUI objects before test cases which cover operations on related GUI objects can be executed. The simple way to build object library is to use the object capture tool provided by RFT to capture a object and then implement key methods of the object if necessary, such as "SetValue" and "VerifyValue" methods in our case. So a prototype of GUI is necessary to build object library in this way, otherwise we have to create object by manual coding. Once a GUI object is stored in object library, RFT can recognize it when a test step operates to it.

In driver functionality part, it implements the translation from test scenarios to executable scripts.

For each test case, the first step is to parse each test step presented by the key-word CNL. Driver retrieves information of application, test suites, test case and then test steps.

During parsing, object mapping is adopted to associate the object described in test step with the one encapsulated in object library. Actually, in test case design phase, a list of GUI objects is provided for user to choose when corresponding objects are already in object library. A mapping rule is defined when adding objects into object library such as name rule. Based on the rule, test driver can find proper object for every step.

Besides object mapping, key-word mapping is also on the way. Since we only have a few key words, it won't be a

```
<Object Class = "java.util.ArrayList">
  <Void Method = "Add">…
    <Object Class = "TestCaseObject">
      <Void Property = "List">
        <Void Method = "Add">
          <Object Class = "TestStepObject">
            <Property = "Value">
              <String>A</String>
            </ Property >
            <Property = "Method">
              <String> SET</String>
            </ Property >
            <Property = "OB">
              <String>UC1_EditBox1</String>
            </ Property >
          </Object >
        </Void>
        ……
      </Void>
    </Object>
  </Void>
  ……
</Object>
```

Fig. 4 Sample XML file

problem to find corresponding methods to set or verify properties. Of course, values are also parsed and assigned in the code.

At last, the driver encodes all the information into a XML file. A stream of test cases are in an array list, and a test case is treated as a Java object which composed of a series of test step objects (Java Object).

Figure 4 shows a piece of an encoding XML file sample for the example in table 2. The "Method" means "Key-Word", "OB" means "Object", and the name for encapsulated object is to add Use Case name as the prefix of the name for object displayed in GUI.

As far as now, test driver completes to interpret test cases into class objects. The final step is to combine configuration information into reformed java codes from XML information for execution.

Due to test driver, test scenarios design can be independent with their execution in an automation procedure. Changes to test case won't introduce manual efforts to update scripts. It is capable for requirement changes. Of course, if GUI objects change, updating to object library is necessary, but it won't cost much.

Another advantage is that test driver makes test case management dashboard adaptable. Since test case management dashboard is loose coupling with the execution engine, if we want to choose another execution tool such as Quick Test Professional, we can keep the test cases in the dashboard but just update the driver.

D. Execution Engine

Execution engine is used to receive scripts from test driver, run the scripts automatically, and record the real-time testing data for automation. Engine should have the ability to compile and execute the code.

In our approach, we choose Rational Functional Tester as our execution engine. Traditionally, RFT is used as a tool to record/replay GUI actions. A limitation is that RFT doesn't support all kinds of applications developed by different programming languages, but it supports Java application well. In our approach, we invoke APIs provided by RFT for automatic execution services.

During the execution, RFT will record all run-time information. And after execution, logs files will be used to identify whether a test is Pass or Fail and notify exceptions when they occur.

E. Reporting Engine

A reporting engine is essential in testing management. It is designed to provide test reports and related functions including providing test coverage information, presenting test results information, exporting reports and so on.

In our solution, executed test cases together with test results of each test cycle will be automatically exported and stored in a repository to match the version of a specific test build once test is finished. It is important from management perspective. Since test cases change as well as code version changes, there should be a way to keep the information that what tests were executed on each specific build.

A popular report which managers use for their decision is about test execution summary. It includes test scope which summarizes how many tests are executed for each component and test results which summarize Pass/Fail status. It adds trouble if we should collect them from logs in a manual way after tests are automatically executed. Our reporting engine can retrieve the information from database on test case level. For test step level, a log analyzer is developed to analyze the information from different logs files.

IV. Discussions and Future Work

As part of our research, a system is developed based on the solution. Till the paper publication, we just complete phase 1 which includes basic functionalities of test case design and management, test execution and general reports. We test our system via a web-based foreign exchange order management system which is developed by Java language. It works well for some basic flows and GUI objects validations. But till now, we didn't apply our automation system in any formal test cycle. As per our plan, the next step will be applying our automation system on GUI functional test in real cases to see its performance.

As far as now, our functional test automation solution on GUI test is presented. Elements of the solution, such as test case management mechanism, key-word driven framework, FRT execution engine and reporting engine, are not novel.

Anyway, it is valuable to combine all the thoughts into one platform in automation practice.

There are several highlights of the solution:

Firstly, the solution supports the test procedure including test case design, test case execution and test reporting. Test management is also one emphasis in our solution including test case management, test scope configuration, test reporting, etc. Also it can reduce the manage efforts on mapping or configuring test cases and executable scripts because there is no visible executable script. So it is not just a solution for test execution automation, but for automating a test procedure.

Secondly, our solution introduces a test driver which is based on the key-word driven framework and a test library. Test driver deals with communication between test cases designed by a key-word CNL and the execution engine RFT. It allows that test cases design can be relatively independent with test execution. It removes the barrier between test automation and testers who don't have programming skills. Then automation programmers can focus on test driver development and in the meanwhile, testers can focus on test case design.

Thirdly, it is capable for changes and easy for maintenance. Once a requirement change comes, if it doesn't require a change on GUI objects, there is no change in the backend but only to update related test cases. Even in case there are changes on GUI objects, what we need to do is only to refine the object library. In case we want to change the support execution engine, test case management dashboard with test cases can still be reused. For maintenance, since test cases are managed structured by the dashboard and there is no visible executable script, it will be quite easy.

There are some challenges in our solution:

Firstly, since we use a key-word control natural language, the definition of Key-word will be a challenge. Currently, we only have three key-words, but it is obviously not enough if we want to adopt our platform to complex GUI clients. Few key-words can make test case design easy but make the object methods capsulation complex. More key-words make mapping key-words to methods easy but make test case design complex. So it is a challenge to decide how many key-words are enough?

Secondly, since we use a controlled natural language which is a semi-formal language, we have to write every action of every object according to the CNL rule. Comparing with natural languages, a test case becomes larger when it using a CNL, so it will cost much more time on test case drafting for testers.

Thirdly, the execution functionality is dependent on the execution engine. So if execution engine doesn't support some kinds of applications, we have to introduce another engine. A new test driver will be developed accordingly. Then can the two drivers work together well when they are attached to the same test case management dashboard?

Based on the discussion above, we can have quite a few works to do in future.

The first thing can be test case automatic generation so that it can reduce the time cost on test case design by the key-word CNL. We can use a scenario-based [7] strategy to design test case and create a graphic builder with some model techniques such as UML or Petri Net [8] to present test workflows on GUI. With proper algorithms [9] [10] [11], test cases and test data can be generated automatically. One challenge is that how to translate test scenarios created by the graphic builder into our key-word CNL so that we don't have to modify our test case management dashboard. In this way, we can make our test fully automated based on graphic test scenarios [12].

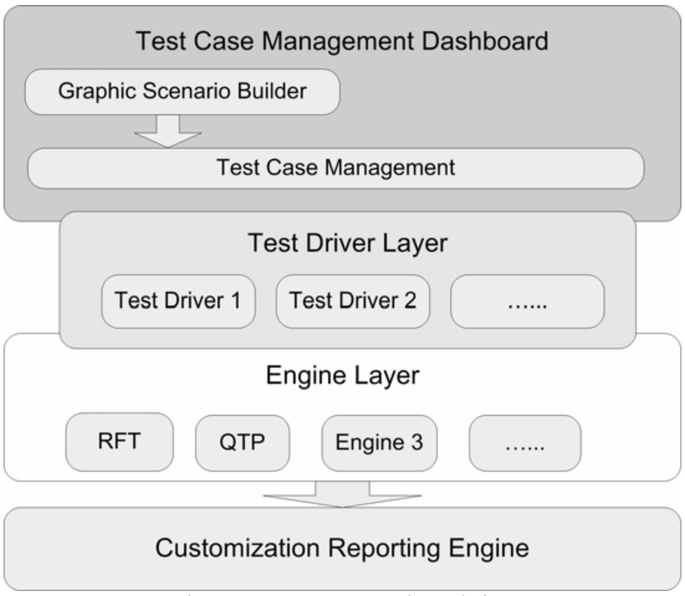

Fig. 4 Future Test Automation Solution

The next thing is to research on multiple test drivers. The goal is to build a layer to manage multiple drivers under a single test case management dashboard so that when user triggers to automate a set of test cases, a proper test driver could be selected to process them and then route executable scripts to a proper execution engine. It's similar with the Enterprise Service Bus (ESB) [13] in a Service Oriented Architecture (SOA) [14].

A third work is to enhance the reporting engine to support customizations to meet requirements from different users.

As a summary, figure 4 describes the vision of our test automation solution in near future. A scenario-driven fully test automation platform on GUI test.

ACKNOWLEDGMENT

The solution comes from test automation practices in State Street Zhejiang University Technology Center (SSTC). Related projects are collaborative efforts between Zhejiang University, China and State Street Corporation, USA. We are thankful for the help from Quality Assurance team of SSTC, as well as the support of State Street Corporation in the research. The contents of this paper are the opinions and conclusions of the authors only and do not necessarily represent the position of State Street Corporation or its subsidiaries, affiliates, officers, directors or employees.

REFERENCES

[1] S.P. Ng, T. Murnane, K. Reed,D. Grant, and T.Y. Chen, “A preliminary survey on software testing practices in Australia”, Software Engineering Conference, 2004. Proceedings. 2004 Australian, 2004 Page(s):116 - 125

[2] Carl J. Nagle, “Test Automation Framework”, 2000, http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm

[3] Nai Yan Zhao, Mi Wan Shum, “Technical Solution to Automate Smoke Test Using Rational Functional Tester and Virtualization Technology”, Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International, Volume 2, Sept. 2006 Page(s):367 - 367.

[4] V. Santiago, A.S. Martins do Amaral, N.L. Vijaykumar, M. de Fatima Mattiello-Francisco, E. Martins, and O.C. Lopes, “A Practical Approach for Automated Test Case Generation using Statecharts”, Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International, Volume 2, Sept. 2006 Page(s):183 - 188

[5] R. Schwitter. “English as a formal specification language”, Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA02),pages 228–232, 2002.

[6] Stephen D. Hendrick, et. al., “Market Analysis: Worldwide Distributed Automated Software Quality Tools 2005-2009 Forecast and 2004 Vendor Shares”, IDC. July 2005.

[7] Carroll, J., “Five reasons for scenario-based design”, Proc.of the IEEE Annual Hawaii International Conference on System Sciences (HICSS). Hawaii, USA, 1999.

[8] James L. Peterson, “Petri Net Theory and the Modeling of System”, Prentice Hall PTR, 1981

[9] Bao-Lin Li, Zhi-Shu Li, Jing-Yu Zhang, and Ji-Rong Sun, “An Automated Test Case Generation Approach by Genetic Simulated Annealing Algorithm”, Natural Computation, 2007. ICNC 2007. Volume 4, 24-27 Aug. 2007 Page(s):106 – 111

[10 Jun-Yi Li, Jia-Guang Sun, and Ying-Ping Lu, “Automated Test Data Generation Based on Program Execution”, Software Engineering Research, Management and Applications, 2006. 09-11 Aug. 2006 Page(s):229 – 236

[11] Jun-Yi Li, Jia-Guang Sun, “Automated Test Data Generation Algorithm Based on Reversed Binary Tree”, Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference, Volume 3, July 30 2007-Aug. 1 2007 Page(s):1124 – 1128

[12] Cheng-hui Huang, Huo Yan Chen, “A Tool to Support Automated Testing for Web Application Scenario”, Systems, Man and Cybernetics, 2006. ICSMC '06. IEEE International Conference, Volume 3, 8-11 Oct. 2006 Page(s):2179 - 2184

[13] Dave Chappell, Enterprise Services Bus, O’Reilly, June, 2004

[14] Thomas Erl, “Part I: SOA and Web Services Fundamentals”, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall, 04 Aug, 2005