Towards Automation in Software Test Life Cycle Based on Multi-Agent

Jingfan Tang
Institute of Software and Intelligent Technology
Hangzhou Dianzi University
Hangzhou, China
tangjf@hdu.edu.cn

***Abstract*—Testing resources are usually involved in the early stage of the SDLC and execute formal Software Testing Life Cycle (STLC), which include test requirement analysis, test cases designing and implementation ,test planning, test execution, defect reporting and analysis, etc. It is expected to spent low cost on software testing process to provide high quality to meet the business requirements. In order to improve the efficiency of the test activities in STLC to address the low cost, it is a trend to automate the test management and execution processes. In this paper, we present an adaptive model of test automation for whole STLC, which is a multi-agent system. It includes Requirement Agent, Construct Agent, Execution Agent, and Report Agent. Through the interaction among the agents, a whole STLC can be executed automatically with efficient management according to the specific requirements.**

Keywords- automation; software test life cycle; muti-agent

I. INTRODUCTION

Quality is critical for the development of software products. And, software testing is regarded as an important way to assure the quality of the software product, which is a process with efficient methodologies, tools and metrics. In order to ensure the quality of the software product produced by DEV, test resources are usually involved in the early stage of the SDLC. The activities in the different stages of the whole software testing life cycle (STLC) include: 1) test requirement analysis; 2) test cases designing and implementation; 3) test planning; 4) test execution; 5) defect tracking; 6) test analysis.

It is expected to spent low cost on software testing process to provide high quality to meet the business requirements. With the development of the software on size and complexity, more and more efforts have to be spent on software testing. But the testing resources and time are usually limited, especially by schedules for market-driven products and in fast-paced software development projects. It has brought big challenge on executing an efficient STLC for software development. In order to improve the efficiency of the test activities in STLC to address the low cost, it is a trend to automate the test management and execution processes.

In this paper, we present an adaptive model of test automation for whole STLC to support the efficient application of the testing framework and tools on different kinds of applications. We implement the architecture as multi-agent system, which includes requirement agent, construct agent, execution agent and report agent.

1) **Requirement agent** is responsible for dealing with the mapping between software requirements and test requirements;
2) **Construct agent** is responsible for generating the test cases according to the test requirements and related test plan for execution;
3) **Execution agent** is responsible for running the test cases under the specific test environment through the specific test tools;
4) **Report agent** is responsible for dealing with the defects reporting and test analysis.

Through the interaction among the agents, a whole STLC can be executed automatically with efficient management according to the specific requirements.

The rest of the paper is organized as following. Section 2 introduces the related work. Section 3 describes the overview of the model which is based on multi-agent. Followed four sections will discuss the multi-agent system in detail. Section 8 concludes.

II. RELATED WORK

There are several types of test automation technology. Test automation categories include: 1) test management, 2) unit test, 3) test cases/data generation, 4) performance test, and 5) functional/system/regression test (i.e. test execution tools) [1].

The most active area should be automated test case and test data generation. For test case generation, the way of virtualized presentation [2] with formalized models tends to be the trend. The State Charts which extend state-transition diagrams with notions of hierarchy, orthogonality and interdependence/ synchronization are used to model software specification [3]. The finite state machine (FSM) also models the various combinations of inputs associated with the user interface of the application to generate test cases randomly [4]. Besides UML, some other new modeling languages such as user community modeling language (UCML) [5] are also used to create visual system usage models. In the meanwhile, many algorithms for automated test data generation are created such as Genetic Simulated Annealing Algorithm [6], algorithm based on

978-1-4244-5392-4/10/$26.00 ©2010 IEEE

Definition-Use-Control (DUC) expression [7], algorithm Based on Reversed Binary Tree [8], etc. Visualization modeling and automated test case generation make it possible that user/analyst can join into the test case design directly.

For test management, an extensible framework called RiOT is presented in [9] to be used to manage testing, and allow dynamic visualization of heterogeneous distributed component-based applications. In [10], it presents a reflective architecture based software testing management model to address the problem of how to effectively manage the whole software testing process. The model integrates reflection theory and software architecture design method, constructed software testing management model by using meta-information theory and meta-model method. Some tools are also developed in industry for test management, such as IBM's Rational Test Manager [11], Mercury's Test Director [12], and TestLink [13] from open source, etc.

For automation of test execution, there have been several approaches in academic research such as data-driven automation test [14], keyword driven automation test [15] and action-driven automation test [16]. The tools are also developed in industry, such as Mercury's Quick Test Professional (QTP) and WinRunner, IBM's Rational Functional Tester (RFT) and Robot [17] on functional testing, and LoadRunner [18] from Mercury, SilkPerformer [19] from Borland, Grinder [20] and JMeter [21] from open source on performance testing.

It is a trend to provide an integrated platform to support the execution of the whole testing process (STLC) automatically with the combination of the automation assets in different environment. Strembeck present an approach [22] for testing functional properties of web applications by using scenarios-based test cases which are generated from use-case diagram of UML. And with meta-data embedded in formal scenario description, a tool transforms scenarios description into executable testing codes, and finally tests the functional properties. And two researchers in China extend the research by extended activity diagram model of UML [23].

In our approach, we use multi-agents to achieve an adaptive model to support the test automation for whole STLC. The term 'agent' used here refers to a relatively simple autonomous system component that exists in a community with other agents and co-operates with them to achieve some complex objectives, such as test automation.

III. MODEL OVERVIEW

We can define our model Am as:

$$Am = (SR, TR, TT, TE, AG)$$

Definition 1: *SR (Software Requirement)* illustrates the requirements of the system to be tested, which can be in the format of functional specification or use cases. It is the original input for our adaptive model which can be defined by a duple (RID, RD), where RID is the unique ID of the software requirement and RD is the description of the software requirement.

Definition 2: *TR (Test Repository)* is a repository to store the test deliverables such as test requirement, test cases, test plan, defects, test log and test reports.

Definition 3: *TT (Test Tools)* illustrates the test tools which will be used for executing the test scripts, such as Rational Functional Tester, Quick Test Professional, LoadRunner and Grinder, etc.

Definition 4: *TE (Test Environment)* illustrates the environment for testing the system with the test tools, which include the configuration of the hardware and software.

Definition 5: *AG (Agent Group)* is defined by *(RqA, CA, EA, RpA)*, where *RqA* denotes requirement agent, *CA* denotes construct agent, *EA* denotes execution agent and *RpA* denotes report agent.

Figure 1 shows the architecture overview.

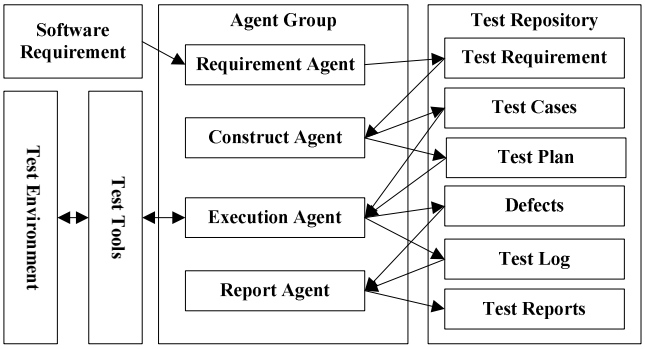

Figure 1. Model overview

When starting the process of test automation for STLC, an agent group is established which includes requirement agent (RqA), construct agent (CA), execution agent (EA) and report agent (RpA). RqA will acquire the software requirement for analysis and generate the test requirement, which can be understood by CA and stored in the test repository (TR). CA will be responsible for constructing the test cases and test plan according to the test requirement. The test cases include the test script and test data, while the test plan identifies the test cases which should be executed as a test suite. EA will be responsible for performing the test plan by running the test scripts under specific test environment through the specific test tools. The defects and test log will be stored into TR during the execution process. Finally, RpA will do the analysis and statistic on the defects and test log to generate all kinds of test report.

IV. REQUIREMENT AGENT

The objective of software testing is to find out the defects in the system through the validation on the software requirement. We will decide what kind of test should be done in the test requirement based on the analysis of the software requirement. In most software projects, the software requirement is illustrated in the functional specification or use cases.

As illustrated in Figure 2, requirement agent (RqA) is responsible for acquiring the software requirement for analysis and generating the test requirement.

In RqA, query model is used to query the software requirement from the functional specification or use cases. Analysis model will query the knowledge from knowledge repository for test requirement analysis. Such knowledge is like some experiences or best practices for helping to do the analysis. The test requirement is generated with the unique ID (TRID) to map with software requirement ID (RID) and stored in TR. For the test requirement, it may cover the test on the functionality, performance and security of the system, which will be decided according to the specific requirement with the guide of the knowledge.

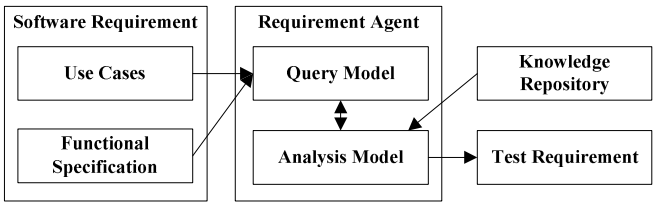

Figure 2. Requirement agent

V. CONSTRUCT AGENT

Construct agent (CA) (see Figure 3) is responsible for constructing the test cases and test plan according to the specific test requirement.

In CA, query model is used to query the test requirement. Some UI can be provided in test design model to develop the efficient test cases. In order to execute the test cases automatically, the test scripts will be generated based on keywords and action words that are considered as best practices in conventional GUI test automation. Instead of producing scripts that can be hard to maintain, we produce sequences of keywords. These sequences are further transformed semi-automatically into Labeled Transition Systems where action words are used as transition labels [24]. Furthermore, the test data will be separated from test scripts to support the data driven testing. The test design model will also establish the mapping between test cases and test requirement through the unique ID (TCID) and TRID. As a test suite, test plan can be used in our model to identify the test cases for different test requirements such as smoke testing, functional testing and performance testing, etc. So, the mapping between test plan and test cases is also established through the unique ID (TPID) and TCID.

The test cases and test plan are also stored in TR as same as test requirement.

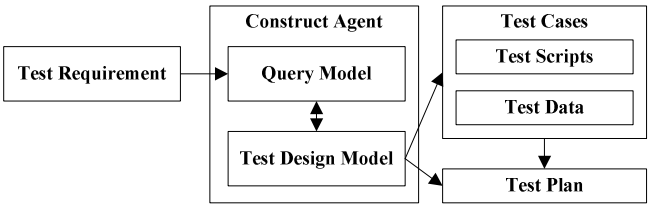

Figure 3. Construct Agent

VI. EXECUTION AGENT

Execution agent (EA) (see Figure 4) is responsible for performing the test plan by running the test scripts under specific test environment through the specific test tools.

In EA, query model is used to query the test plan for execution. A new session will be established for the specific test execution since the test plan may be executed for many times. The mapping between the test execution and test plan will be established through the unique ID (TEID). With the open interfaces provided by the test tools such as Rational Functional Tester, Quick Test Professional, LoadRunner and Grinder, the test execution model can route the execution of the test scripts on the specific test tools automatically to support the keyword driven testing.

During the test execution, the defects will be recorded for tracking in some defect tracking tools, such as Rational ClearQuest, Bugzilla and JIRA, etc. The mapping between defects and test cases in the execution of test plan will also be established through the unique defect ID. That means a defect (defect ID) is found by running a specific test case (TCID) in a round of test execution (TEID) of a test plan (TPID).

A test log will be generated after the test execution. It may include the information of the test date, tester, executed test plan and test cases, and test efforts, which can be used for test analysis. The defects and test log are also stored in TR.

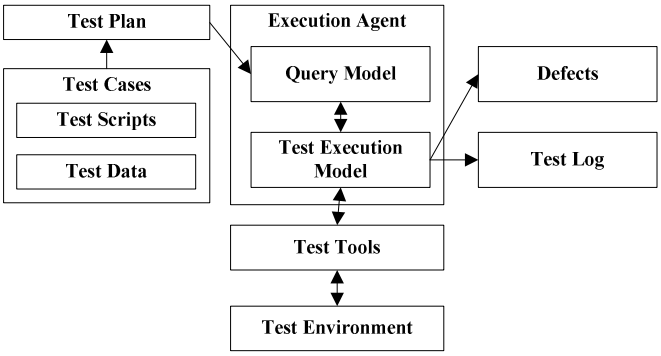

Figure 4. Execution Agent

VII. REPORT AGENT

Test analysis is very important for software testing. An excellent testing process not only includes finding the defects but also providing the analysis on the defects. Through the result of the test analysis, we can find the root cause of the defect, prevent the occurrence of the defect and improve the quality in future.

In our model, report agent (RpA) (see Figure 5) is responsible for generating all kinds of test reports according to the defects and test log.

In RpA, query model is used to query the data of defects and test log, which are stored in TR in the format of data file or DB. We can define the quality metrics and templates through the UI provided by report model. As long as the data can be provided from the defects and test log, the related test reports

can be generated, such as defect report, test summary and test analysis report, etc.

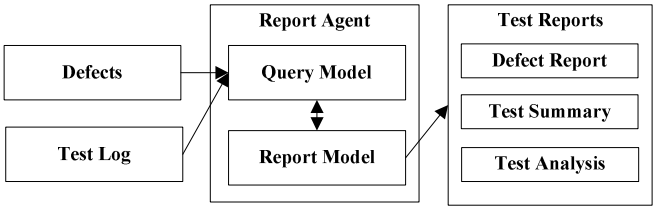

Figure 5. Report Agent

VIII. CONCLUSION

Software testing plays a key role in software development as software quality becomes more and more critical. Test automation is a valuable area to reduce manual efforts and make test efficient. In this paper, we present an adaptive and multi-agent model of test automation for the whole STLC which include test requirement analysis, test cases designing and implementation ,test planning, test execution, defect reporting and analysis, etc. It is also a flexible framework to support the integration with the different test tools. Through the interaction among the agents, it can automate the test processes to reduce the cost and improve the efficiency.

REFERENCES

[1] Tom Wissink and Carlos Amaro, "Successful Test Automation for Software Maintenance", Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM'06).

[2] 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, pp. 367 – 367, Sept. 2006.

[3] 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, pp. 183 – 188, Sept. 2006.

[4] J. Alava, T.M. King, and P.J. Clarke, "Automatic Validation of Java Page Flows Using Model-Based Coverage Criteria", Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International, Volume 1, pp. 439 – 446, Sept. 2006.

[5] Scott Barber, "User Community Modeling Language (UCMLTM) v1.1 for Performance Test Workloads." May 2004, StarEast.

[6] 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, pp. 106-111, Aug. 2007.

[7] Jun-Yi Li, Jia-Guang Sun, and Ying-Ping Lu, "Automated Test Data Generation Based on Program Execution", Software Engineering Research, Management and Applications, pp. 229 – 236, Aug. 2006.

[8] 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, pp. 1124 – 1128, July 30 2007-Aug. 1 2007.

[9] Sudipto Ghosh, Nishant Bawa, Gerald Craig, Ketaki Kalgaonkar, "A Test Management and Software Visualization Framework for Heterogeneous Distributed Applications", Proceedings of the 6th IEEE International Symposium on High Assurance Systems Engineering (HASE'01).

[10] YAO Jun-feng, YING Shi, LUO Ju-bo, XIE Dan, JIA Xiang-yang, "Reflective Architecture Based Software Testing Management Model", IEEE MIT 2006, pp. 821-825.

[11] Rational Test Manager, http://www-306.ibm.com/software/awdtools /test/manager/.

[12] TestDirector, http://www.starbase.co.uk/mercury-test-director.asp.

[13] TestLink, http://testlink.org/wordpress/.

[14] Zambelich, K., "Totally Data-Driven Automated Testing", 1998, http://www.sqa-test.com/w_paper1.html.

[15] Carl J. Nagle, "Test Automation Frameworks", 2002, http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm.

[16] Li Feng, Sheng Zhuang, "Action-Driven Automation Test Framework for Graphical User Interface (GUI) Software Testing", Autotestcon, 2007 IEEE, pp.22-27.

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

[18] Mercury Interactive Corporation, "Load Testing to Predict Web Performance" Technical Report WP-1079-0604, Mercury Interactive Corporation, 2004.

[19] "Borland® SilkPerformer® data sheet". Retrieved July 23, 2007, from http://www.borland.com/resources/en/pdf/ products/silk/silkperformer_datasheet.pdf.

[20] Philip Aston, "The Grinder, a Java Load Testing Framework", (2007, March 18). Retrieved July 23, 2007, from http://grinder.sourceforge.net/.

[21] "Apache JMeter", (1998, December 15). Retrieved July 23, 2007, from http://jakarta.apache.org/jmeter/index.html.

[22] M. Strembeck, U. Zdun, "Scenario-based Component Testing Using Embedded Metadata", Proc. of the Workshop on Testing of Component-based Systems (TECOS), Lecture Notes in Informatics (LNI), Erfurt, Germany, September, 2004.

[23] 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, pp. 2179 – 2184, 8-11 Oct. 2006.

[24] Mika Katara, Antti Kervinen, Tuula Paakkonen, Mikko Satama, "Towards Deploying Model-Based Testing with a Domain-Specific Modeling Approach", Proceedings of the Testing: Academic & Industrial Conference – Practice and Research Techniques (TAIC PART'06),pp.81-89.