Cooperative Agent Approach to Quality Assurance and Testing Web Software

Hong Zhu
Dept. of Computing, Oxford Brookes University
Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk

Abstract

This paper applies Lehman's theory of software evolution to analyse the characteristics of web-based applications and identifies the essences and incidents that cause difficulties in developing high quality web-based applications. It is argued that they belong to Lehman's E-type systems, hence satisfy Lehman's eight laws of software evolution. The uncertainties underlying the development of web applications are analyzed and their implications are discussed. In order to support sustainable long term evolution of such systems, we proposed a cooperative multi-agent system approach to support both development and maintenance activities. A prototype system with emphasis on testing and quality assurance is reported.

The Web has become a distributed and hypermedia software platform. It has stimulated many new applications [1]. However, developing web applications are complex, difficult and expensive. As a consequence, software quality suffers most. In this paper, we discuss why testing web applications and quality assurance are difficult and propose a cooperative agent approach to overcome the problems.

1. Characteristics of Web Applications

The causes of difficulties in software development can be essences or incidents [2]. Hence, it is desirable to understand the essence and incident factors that affect web application development.

1.1. Classification of software systems

According to Lehman [3], software systems can be classified into three types according to what 'correctness' means to the particular system. An *S-type* program is required to satisfy a pre-stated specification. For such a system, correctness is the absolute relationship between the specification and the program. Many safety critical applications belong to this type. A *P-type* program is required to form an acceptable solution to a stated problem in the real world. The correctness of a P-type program is determined by the acceptability of the solution to the stated problem. An *E-type* program is required to solve a problem or implement an application in a real-world domain which often has no clearly stated specification. The correctness of such a system is judged by the users. Obviously, many kinds of web applications such as e-commerce, e-government, e-science, etc., belong to the E-type.

1.2. Uncertainties underlying web applications

Different types of systems in Lehman's classification demonstrate different evolution processes because they are affected by different types of uncertainties.

Generally speaking, there are three types of uncertainties associated with software development [4]. *Gödel-like uncertainty* arises because software systems are models. The representation of such a model and its relationship to the real world is Gödel incomplete. Consequently, the properties of a program cannot be completely known. *Heisenberg-type uncertainty* results from the processes of using the system, which inevitably change the user's perception and understanding of the application. A common phenomenon in the development of software systems is that the users are uncertain about the requirements, but they often insist that '*I'll know it when I see it*' [5]. Uncertainties from this source exhibit themselves in the form of changing requirements. *Pragmatic uncertainty* is due to the human participation in the development process. Software development process is still a manual activity during which errors are made and faults are introduced. Many types of risks in software development are due to this type of uncertainty. For example, the adaptation of a new development method and the use of a new software tool or programming language may introduce uncertainty to the quality of the product and the development process.

Although these sources of uncertainties are associated with all software development projects, their impacts on web applications are much more serious. Being E-type systems, web applications are judged by users' acceptances for their correctness. Many web applications are novel to the majority of users. Users are bound to change their perception and understanding of the application. For example, many web-based e-banking systems have been developed based on knowledge about how people use banking facilities via accesses to local branches of banks and via telephone banking facilities. Assumptions are made about how people's habits and behaviours can be supported by and adapted to online accesses through the Internet. Once an e-banking system is implemented and put into operation, the users' understanding of e-banking systems and their way of banking started to change. New requirements emerged as the results of using such systems. For example, many banks in UK have changed their terms and conditions of opening and using bank accounts in the past a few years to meet the needs of online banking. Consequently, modifications of the system must be made to meet users' new requirements. At the time of developing the first version of e-banking systems, it was unpredictable what requirements would emerge before actually implementing and using them. The same can be said to the requirements to emerge in the future. Since Heisenberg-type uncertainties always play a significant role in the development of E-type systems, it will continue to be a major uncertainty in the development of web applications.

Gödel-like uncertainty also hampers the development of high quality web applications. On one hand, the development of a web application heavily depends on the existence of an accurate model of the real world as the basis of the design and implementation of the system. On the other hand, there is few mature theory and methods that helps developers to build a good model of the system and environment of web applications. The complexity of web applications is inevitable because they execute on distributed, hypermedia, heterogeneous computer platform. Moreover, they are open to the environment, hence vulnerable to malicious attacks. They are often depending on unreliable and uncontrollable hardware and software resources; hence they have to be fault tolerant and cooperative to other systems, etc. Existing formalisms and semi-formal notations, such as UML, become powerless to handle all these issues at the same time. Although such uncertainties may gradually diminish as research and technical development progresses in the long term, there seems no immediate solution but to integrate existing techniques and research results.

Pragmatic uncertainty also contributes to the difficulties in the quality assurances and testing web applications. It is because web technology has been rapidly developed in the past a few years. A large number of new techniques have emerged. It is predictable that more web techniques in the laboratory will become available for practical uses in the next a few years. Moreover, in the past a few year, a large number of persons have entered the IT profession. Many of them have limited experience and training in software development. These are the main sources of uncertainties of the pragmatic type. Such causes of difficulties are incidents rather than essences. For a long time, software quality assurance as well as verification, validation and testing methods and techniques have been developed to reduce the pragmatic uncertainty and to minimize their impact on software development. Principles proved to be effective for software testing and quality assurance should be equally applicable to the web applications. Existing methods, techniques and tools can be adapted to the new web technology. Yet, new methods and techniques must also be developed for deal with the novel features of web applications.

1.3. Laws of evolution

How can we overcome the difficulties? Lehman studied a number of systems that had survived in a long process of evolution and proposed a set of laws of E-type software systems [3], which is quoted in Table 1 below. In addition to Lehman's laws, we also observed a common phenomenon of web-based systems in our investigation of web-based applications. That is, in a web-based system, there are almost always components developed using different methods, such as code written in different languages and/or executing on different platforms, data represented in different formats, interfaces interacting with different (versions of) external software systems using different protocols, etc. This is called the *law of diversity*, which is also listed in Table 1.

Lehman's laws can be considered as a 'survival guide' for the evolutionary development of E-type systems. Violating Lehman's laws in the development of an E-type system may well mean a death penalty. Here, the death of a software system should be understood in Peter Naur' sense [6], that is, the state of death becomes visible when demands for modifications of the program cannot be intelligently answered although the program may continue to be used for execution and provide useful results. These laws provide the clues for how software development processes should be organized and how supporting tools should be constructed. They imply the following high level requirements on the quality assurance and testing activities in web application development.

First, to develop a high quality E-type systems, and web applications in particular, an evolutionarily development strategy must be adopted. Moreover, quality assurance and testing, verification and validation techniques must fit into this evolution process.

Second, to enable a long term sustainable evolutionary development process, it is vital to prevent the system's complexity increasing and quality declining out of control. This requires constantly maintaining a good knowledge about the system.

Third, it is essential to build feedback loops between the users and the developers. The developers must commit a continuous effort to adapt the system to meet users' new requirements.

Finally, various testing, verification and validation tools must be integrated into one environment to deal with the diversity of the components and their execution platform and environment. The environments must also be easily extended so that new tools can be integrated in the future as new development techniques emerge in the future.

Table 1 Laws of Software Evolution

Law	Description
Continuing Change	E-type systems must be continually adapted else they become progressively less satisfactory in use.
Increasing Complexity	As an E-type system is evolved its complexity increases unless work is done to maintain or reduce it.
Self Regulation	Global E-type system evolution processes are self regulating.
Organizational Stability	Unless feedback mechanisms are appropriately adjusted, average effective global activity rate in an evolving E-type system tends to remain constant over product lifetime.
Conservation of familiarity	The incremental growth and long term growth rate of E-type systems tend to decline.
Continuing Growth	The functional capability of E-type systems must be continually increased to maintain user satisfaction over the system lifetime.
Declining Quality	The quality of E-type systems will appear to be declining unless they are rigorously adapted, as required, to take into account changes in the operational environment.
Feedback System	E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems.
Diversity	An E-type system contains components that are developed and integrated into the system using a diversity of techniques.

2. The Proposed Approach

To satisfy these requirements, we proposed a cooperative agent approach to constructing quality assurance and testing environment for web applications [7~9]. The basic ideas can be summarized as follows.

The software environment consists of the two types of agents. *Service agents* provide various supports to the development of software systems in an evolutionary strategy. They fulfil the functional requirements of development and quality assurance and testing, verification and validation functionalities. *Management agents* manage service agents and responsible for the registration of agents' capabilities, task scheduling, and monitoring and recording agents' states and the system's behaviours. Each service agent is specialized to perform a specific functional task and deal with one representation format. They cooperate with each other to fulfil more complicated tasks. Various development and quality assurance techniques and tools can thus be nicely integrated. Boundaries between different representation formats and notations can be bridged through cooperation between agents.

These agents co-exist with the application software system throughout the application system's whole lifecycle to support the modifications of the system. They monitor the evolution process and record the modifications of the system and the rationales behind the modifications. They extract, collect, store and process the information about the application system and its performance, and present such knowledge to human beings or other software tools when requested. They interact with the users and developers cooperatively. The environment grows with the application system as new tools are integrated into the environment to support the development and maintenance of new components and as the knowledge about the system is accumulated over the time. The relationship between the tools and the system is similar to the relationship between a tree and its natural environment where it is growing, and a between a human and his/her social environment that changes as the person is growing up. Because of these features, we call such a software environment a *growth environment*. It significantly differs from software development environments and run-time support environments such as middleware, where evolution is not adequately supported.

In order to enable agents to cooperate effectively with each other and with human users, they communicate with each other through a flexible and collaboration protocol and codify the contents of messages in an ontology which represents knowledge about the application domain and software engineering. The use of ontology also enables high extendibility of the system so that agents can be easily added into the system, removed from the system, or upgraded by new versions.

3. Prototype System

To demonstrate the feasibility and capability of the above proposed approach, we designed and implemented a prototype growth environment for quality assurance and testing web-based applications.

As shown in Figure 1, the environment consists of a number of agents to fulfil testing related tasks for web-based applications. These agents can be distributed to different computers of application servers, test servers and clients. Table 2 briefly describes the agents that have been implemented; see [8] for more details.

An ontology of software testing is developed and represented in XML for the communications between agents [8,9,10]. The interaction protocol is developed on the basis of speech-act. The use of ontology enables agents to communicate with each other and with human users at a very high level of abstraction.

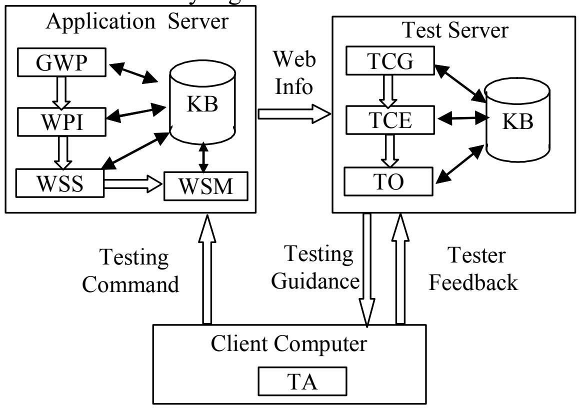

Figure 1 System Structure

Table 2 Agents for testing web applications

Agent	Functionality
GWP – Get Web Page	Retrieve web pages from a web site.
WPI – Web Page Information	Analyse the source code of a web page, and extract the page title, metadata, hyperlinks and structural information from the code.
WSS - Web Site Structure	Analyse the hyperlink structure of a web site, and generate a node-link-graph describing the structure.
TCG - Test Case Generator	Generate test cases to test a web site according to certain testing criteria.
TCE - Test Case Executor	Execute the test cases, and generate execution results.
TO – Test Oracle	Verify whether the testing results match a given software specification.
TA - Testing Assistant	Perform as user interface and guide human testers in the process of testing.
WSM- Web Site Monitor	Monitor the changes of web sites, and generate new testing tasks accordingly.

4. Conclusion

The application of Lehman's theory of software evolution to web-based applications shows that they are by nature evolutionary and, hence, satisfy Lehman's laws of evolution. The essence of web applications implies that supporting their sustainable long term evolution should play the central role in developing quality assurance and testing techniques and tools. Therefore, two basic requirements of such a software environment can be identified. First, the environment should facilitate flexible integrations of tools for developing, maintaining and testing various kinds of software artefacts in a variety of formats over a long period of evolution. Second, it should enable effective communications between human beings and the environment so that the knowledge about the system and its evolution process can be recorded, retrieved and effectively used for future modification of the system. Our solution to meet these requirements is a cooperative multi-agent software growth environment. In this environment, various tools are implemented as cooperative agents interacting with each other and with human users at a high level of abstraction using ontology. We have designed and implemented a prototype system for testing web applications. Preliminary experiments with the prototype have shown some promising results.

Acknowledgement. The project reported in the paper is partly supported by Oxford Brookes University Research Bursary and China High Technology Research Programme under grant 2002AA116070. The author is grateful to Mr. Qingning Huo who designed and implemented the prototype system reported in this paper.

References

[1] Crowder, R., Wills, G., and Hall, W. Hypermedia information management: A new paradigm. Proc. of 3rd Int Conf on Management Innovation in Manufacture, July 1998, 329-334.

[2] Brooks, F. P. Jr, No silver bullet: essence and accidents of software engineering, IEEE Computer, 1987, pp10~19.

[3] Lehman M. M. and Ramil, J. F. Rules and Tools for Software Evolution Planning and Management. Annals of Software Engineering, Special Issue on Software Management, 11(1) 2001, 15-44.

[4] Lehman, M. M. Uncertainty in Computer Application. C.ACM, 33(5), 1990, 584-586.

[5] Boehm, B. Requirements that Handle IKIWISI, COTS, and Rapid Change. IEEE Computer, July 2000, 99-102.

[6] Naur, P. Programming as theory building, Computing: A Human Activity. ACM Press, 1992, 37-48.

[7] Zhu, H., Greenwood, S., Huo, Q. and Zhang, Y., Towards agent-oriented quality management of information systems, Workshop Notes of 2nd International Bi-Conference Workshop on Agent-Oriented Information Systems at AAAI'2000, Austin, USA, July 30, 2000, 57-64.

[8] Huo, Q., Zhu, H., and Greenwood, S., A Multi-Agent Software Environment for Testing Web-based Applications, Proc. of COMPSAC'03, Dallas, 2003, 210-215.

[9] Huo, Q., Zhu, H. and Greenwood, S. Using Ontology in Agent-based Web Testing. Proc. of ICIIT'2002, Sept., 2002, Beijing, China.

[10] Zhu, H. and Huo, Q., Developing A Software Testing Ontology in UML for A Software Growth Environment of Web-Based Applications, To appear in Software Evolution with UML and XML, Hongji Yang (eds.), Idea Group Inc.