行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※
※ ※
※一支援 3-TIER PROGRAMS 測試的工具之研究與製作 ※
※ ※
※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：V 個別型計畫　　□整合型計畫
計畫編號：NSC 89 –2213 – E – 009 – 010
執行期間： 88 年 8 月 1 日至 89 年 7 月 31 日

計畫主持人： 王豐堅
共同主持人：

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

執行單位：國立交通大學資訊工程系

中 華 民 國 90 年 2 月 26 日

行政院國家科學委員會專題研究計畫成果報告

計畫編號：NSC 89 –2213 – E – 009 – 010

執行期限：88 年 8 月 1 日至 89 年 7 月 31 日

主持人：王豐堅　國立交通大學資訊工程系

計畫參與人員：楊基載等研究生

一、中文摘要

在程式發展階段利用軟體測試技術來確保應用程式的品質，是開發高品質軟體的重要一環。目前大量採用資料庫管理系統(DBMS)**之商業應用環境，幾乎都以** WWW **瀏覽器與跨平台的** Java **為基礎，發展出新的應用程式架構。這種獲得** Oracle、IBM DB2、Microsoft SQL server **等主流資料庫系統支援的「三層式應用程式架構」** (Three-Tier Application Architecture)**，係採用** WWW **瀏覽器或其他** GUI **程式作為客戶端**(client)**，連往前端處理伺服器**(front-end pre-process server) **，再透過開放式資料庫連線**(**如** ODBC)**連往後端的資料庫系統伺服器**(back-end DBMS server)**。**

我們在本計劃裏以 ANSI/IEEE **的軟體測試標準為基礎，為測試者規劃一套適合三層式架構的軟體測試流程與環境，製作出一套半自動產生測試資料**(test case)**產生器以輔助測試者進行** GUI **介面、主從式架構、三層式架構整合，也製作一評估整個流程的效能**(effectiveness)**及效率**(performance)**分析器。**

關鍵詞：網際網路, 軟體測試、軟體可重覆使用性、三層式軟體架構

1. Abstract

Web model and its related improvement give Web application designers flexibility at choosing proper development products for their implementation. Current developers of large Web application do not have sufficient and powerful tools to debug or test their Web applications. [1] addresses the necessity of software testing support to handle the complexity of Web applications. Existing Web testing tools on Internet are usually made for verifying the syntax in HTML documents, checking the hyperlink integrity in a set of HTML documents, testing GUI components embedded in browsers, and measuring the performance of the Web application. Few products support overall Web applications testing [2]. [3] and [4] test the software components such as Java Applet and ActiveX objects which are embedded in the Web pages. [5] extends traditional GUI testing tools to test the GUI events inside the Web browsers. [6] helps justify the result shown on the Web browser's window by matching text patterns or pixel-level comparison. [7] checks the documents for syntax and compatibility to popular Web browsers.

The article presents a software architecture that integrates several conventional testing tools. The architecture extends these traditional software testing architectures and software patterns [8] [9] to ease the description and design of Web-application testing tasks. The integration of Web-testing components can reduce the insufficiency of independent Web testing tools mentioned above for complicated Web application testing. We also constructed a testing environment for Web applications to demonstrate the reuse of the software architecture. With object-oriented technique, the architecture itself provides a clear picture of software components for reuse including tool reuse, architecture reuse, etc.

Keywords: Web-application testing, software architecture, framework reuse

2. Constituents of Web Applications

Web model provides application platforms or application designers with several locations to place code for Web computation and alternative mechanisms to solve particular missions.

Figure 1 depicts the typical constituents of the Web application.

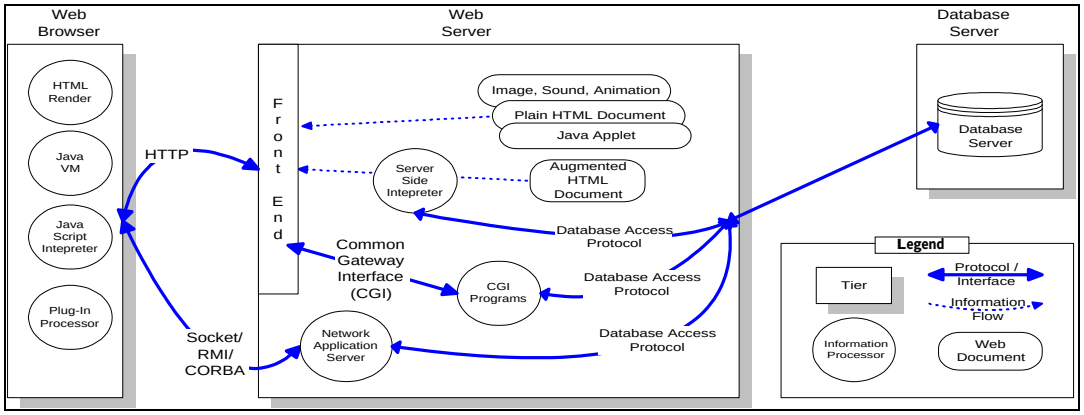

Figure 1. Constituents of typical Web applications

Contents of Web applications are usually stored on the Web server or database server. They can be static

HTML documents, image files, video files, or even kinds of programs run on Web servers or Web clients. Information processor in Web application model accepts incoming requests and returns *contents*. The request is either processed and returned directly, or translated and delegated to another information processor prior to return.

The *Web browser* is capable of retrieving hyper-text documents, as requested by the application users, from the Web server via HTTP protocol. It renders the hyper-text document in HTML (Hyper-Text Markup Language) format on the screen. Contemporary Web browsers also embed Java virtual machine and Java Script interpreter to execute the Java Applets or Java Scripts specified in the documents. Additional information processor such as Netscape Communicator's *plug-in* modules and Microsoft Explorer's ActiveX objects, which are browser-loadable software modules, can extend browser's functionality.

HTTP daemon is placed at the Web server to accept the HTTP requests from the browsers. According to Web server's configuration, it may forward the request to (1) document retriever for serving stored HTML documents, Java Applets, or multimedia files, or (2) to other information processor on the Web server, such as CGI programs for dynamically generated HTML documents and contents. Web servers are sometimes equipped with information processor, e.g. Apache Module, or Active Server Page Engine to perform the computation defined in augmented HTML documents before sending them to browsers.

HTTP-cookie is an entity issued by the information processor on the Web server and sent to Web browser via HTTP protocol. HTTP-cookie is stored at Web browser side, and is sent back to the Web browser conditionally to inform the information processor on the Web server. Cookie is mainly used to make stateless HTTP transactions stateful. Temporary information which need to be kept during consecutive HTTP transactions can also be stored in the HTTP-cookie. Protocols convey command, document or executable between *information processor*. *Hyper-Text Transfer Protocol, or HTTP*, is used for communication between Web browsers and Web servers. *The Common Gateway Interface, or CGI*, is a standard for external gateway programs to interface with information servers such as HTTP servers.

The placement of constituents in Web model can divide Web application constituents into three major tiers: Web browser tier, Web server tier, and database server tier. The information process in the application is passed through each tier. The user interaction is performed at the Web-browser tier. The program logic computation is performed at the Web server tier. The database operation is done at the database-server tier. Hence, the Web application model is also known as a *three-tier* application architecture. When the database server tier is omitted, it is known as a *two-tier* application model.

3. Domain Components for Web Application Testing

Although Java applets are very popular in Web application, currently the domain components for Web application testing are not concerned much. One reason to the omission is that testing tasks of Java applets performed in present Web application testing products are mainly related to its Window application testing aspects, instead of less explored Web application testing aspects. Java applets can be viewed as a platform-independent Window application running on Web browsers with special restriction on network connection destination and local file system access.

To perform Web application testing with respect to above application scenarios, domain components might be included to testing environment as the primitives, to help describe testing tasks. Tools perform Web application testing tasks can be composed by selecting proper domain components from Web-enhanced software testing architecture. For example, a tool which tests Web application written in ASP scripts may choose to use *Script-Side Script Analyzer* to fetch script source and analyze the control flow and data flow within the scripts. The information can be passed to traditional domain components to generate test cases. The test cases are then transformed into the format known by *Test Case Executor* to describe the execution steps of Web applications (i.e. Mouse click on Web link, image map; data values filled in HTML form) and stored in *Test Suite/Case Repository*. During the test case running time, with the help of the testing execution component such as *Form Filler* and *GUI Event Generator*, tester can run test cases by confirming each operation predefined in the test script without performing the operation personally. The fail or pass of a test case can be manually judged by the testing staffs or automatically judged by the *Test Oracle*. *Test Coverage Analyzer* and related reporting components in section 4 then give the testing report on the percentage of tested part and outcome of each test case.

4. Architecture of Web Testing Environment

[10] proposed an architecture for traditional software testing environments, and it is well evaluated in [11]. We extend this architecture for testing of Web applications testing as described in this section.

4.1 The Architecture

According to the architecture in [10], a software testing environment consists of five subsystems. With the growth of Web application techniques, more and more Web programming styles (e.g. ASP, JavaScript) have been proposed. These programming styles introduce several new techniques which were not used in conventional software. For example, one document may contain several code fragments written in

different programming languages, and these fragments may be interpreted in different tiers such as browser, server, database, …etc. Therefore, one Web application should be analyzed at browser, server, even database tire, and the corresponding analysis services for different programming languages are also needed. We add a new subsystem named Source Document Analysis into above architecture to handle the testing problems introduced by these new programming styles. Figure 2 shows an overview of this architecture, where solid lines indicate data flow.

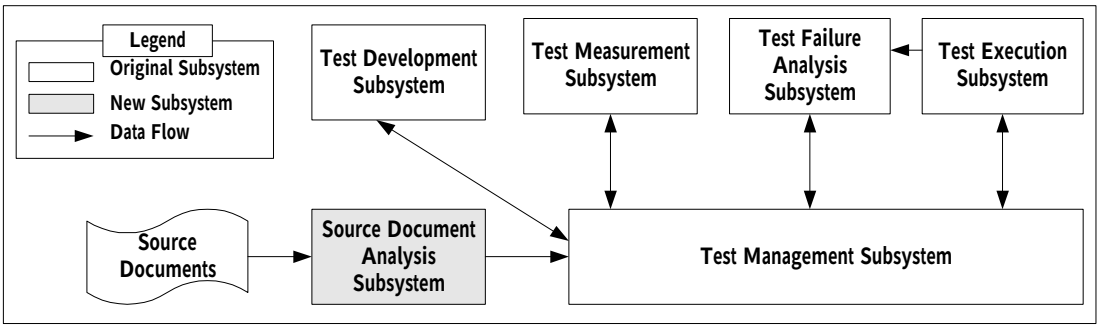

Figure 2. Architecture of Web Application Testing Environment

In this architecture, Source Document Analysis Subsystem (SDAS) is used to analyze the source documents, and extract some useful information such as control flow model. Test Management Subsystem (TMS) serves as a warehouse which stores all extracted information, and provides the access interfaces for other subsystems to manipulate stored information. Test Development Subsystem (TDS) provides the maintenance functions for testers to create, modify, and delete test cases, which are stored in TMS. Test Execution Subsystem (TES) is used to execute test cases, activate Web application with designated paths, fill corresponding test data, and capture the execution results. Test Failure Analysis Subsystem (TFAS) verifies test cases by analyzing the captured execution results to determine whether these test cases match the specification of Web applications, and sends the result of verification to TMS. TFAS also summarize all verification result of all test cases to show how many test cases are executed, verified, …etc. Test Measurement Subsystem (TMS) measures whether and how much of a test criterion is adequately satisfied.

4.2 Source Document Analysis Subsystem, SDAS

Different programming approaches applied for Web application developments have different characteristics. For example, server side programming is focused on database accesses and able to generate documents to Web browser according to the result of database queries. Client side programming is focused on GUI representation and manipulation in Web browser. A programming approach may need a distinct programming language, which is associated with a set of tools, such as *Server-Side Script Interpreter*, *Client-Side Script Interpreter*, *HTML Analyzer*, and so forth.

All of these tools designed to analyze the source documents and extract some information (e.g. hyperlink) are called *Analyzer*. There are two categories of *Analyzer*s for Internet software developed, *HTML Analyzer* and *Script Analyzer*. *HTML Analyzer* processes HTML fragments in source documents to extract information such as CGI From. *Script Analyzer* is used to process the embedded script fragments and extract information such as control flow of these script fragments. *Script Analyzer* is divided into two categories, *Client-Side Script Analyzer* and *Server-Side Script Analyzer*, according to the location which the script fragments are interpreted. *Client-Side Script Analyzer* handles the script fragments which are interpreted in Web browser, and *Server-Side Script Analyzer* handles the script fragments which interpreted in Web server.

SDAS extracts information such as control flow from source documents, and sends them to TMS. Control flow is useful in software testing. [12] and [13] proposed methods to construct control flow of Web applications based on the hyperlink relations between source documents. A source document may contain HTML, server-side script, and client-side script at the same time. Analyzers are designed to extract hyperlinks in three parts respectively. Figure 3 shows an architecture of SDAS to construct the control flow of Web applications. The analysis process contains following steps:

1. Source documents are sent to *Server-Side Script Analyzer* to analyze the server-side script (i.e. ASP) fragments, and are sent to *Server-Side Script Interpreter* to interpret the server-side script fragments.
2. The interpreted source documents are sent to *Client-Side Script Analyzer* to analyze the client-side script (i.e. JavaScript) fragments, and are sent to *Client-Side Script Interpreter* to interpret the client-side script fragments.
3. The interpreted source documents from step 2 are sent to *HTML Analyzer* to analyze the HTML fragments.

All extracted hyperlinks from step 1,2,and 3 are sent to *Control Flow Builder* to construct the control flow of the Web application, which is then stored in TMS.

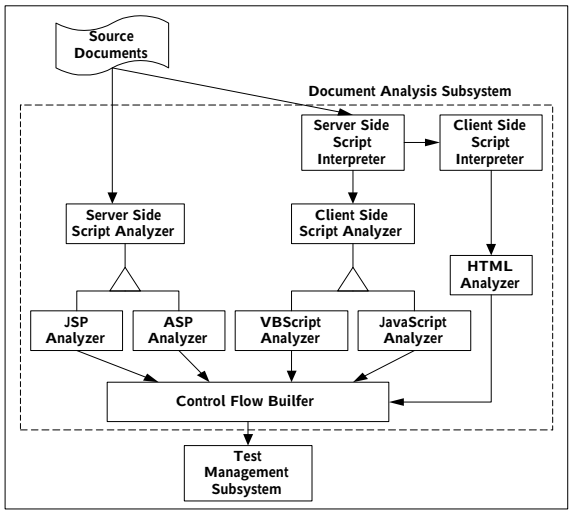

Figure 3. Source Document Analysis Subsystem

4.3 Test Management Subsystem, TMS

Testing (and validation) deals with many artifacts which may be created during earlier development phase(s) or even validation phase [10]. Compared with traditional software, Web applications involve additional roles for Web such as Web server and browser, and additional control mechanisms such as cookie and session. Therefore, testing on Web applications is different (and might be more complicated than) on traditional software, and the test artifact management (e.g. manipulation of test cases) is more important. TMS works as the warehouse of other subsystem to provide testing artifact management. It contains *Application Information Repository* and *Test Suite/Case Repository*, where each repository has its own manager to handle repository manipulation.

A testing model ([13]) is used to describe some behavior of Web applications and the corresponding information such as control flow and data flow is stored in *Application Information Repository*. *Test Suite/Case Repository* stores test suites (and cases) which include test data, execution path, execution result, test report, and so forth. TMS provides a set of repository access interfaces and separates the interfaces from their implementations. With these access interfaces, other subsystems can create, manipulate, delete and query Repositories without concerning their implementations.

In TMS, Manager pattern [9] is applied to achieve this goal. To manipulate test suites or cases, the client subsystem sends request messages to *Test Suite/Case Manager*, which then loads or creates the *Test Case* objects correspondingly. After manipulation, client subsystem can use the Test Case object directly. Since the client subsystem is not related to the implementation of *Test Suite/Case Repository*, the former does not need modification when testers change the implementation of the latter. The implementation of *Application Information Manager* is similar to that of *Test Suite/Case Manager*.

5. Conclusion and Future Work

In this article, we designed and constructed a reusable Web application testing environments by extending a well-evaluated architecture and applying some design patterns. The architecture contains six subsystems, and the testing processes (e.g. test case generation) can be achieved with the cooperation of these subsystems. To demonstrate the usability of this architecture, a prototyped Web-application testing environment was built. The corresponding classes implmented are put in our labrotary and introduced in [13].

Although many facets in Web applications are discussed, there are several popular ones not mentioned. For example, many Web applications such as online ordering system are associated with a database, and many users' behaviors will cause the database accesses. We are now focusing on these issues and propose a set of components to handle the interactions between Web server and database.

Reference:

[1] Fromme B., "*Web Software Testing – Challenges and Solutions*", InterWork'98 Conference, 1998.

[2] "*Testing and Testing Management Tools*", in http://www.methods-tools.com/tools/testing.html.

[3] Sun Microsystems, "*SunTest Suite*", in http://www.sun.com/suntest.

[4] Softbridge Inc., "*Web Analyst*", in http://www.softbridge.com.

[5] Mercury Interactive Corp., "*Powerful Test Automation for Enterprise – Mecury Interactive's WinRunner*", in http://secure.merc-int.com/products/winrunner5/.

[6] Mercury Interactive Corp., "*Visual Web Site Management – Mecury Interactives's Astra SiteManager*", in http://www.merc-int.com/products/astrasmguide.html.

[7] Rational Software, "*Visual Test 4.0 White Paper*", in http://www.rational.com/products/visual_test/prodinfo/whitepapers/dynamic.jtmpl?doc_key=100464

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison-Wesley, 1994.

[9] Reboert Martin, Dirk Riehle, and Frank Buschmann, *Pattern Languages of Program Design 3*, Addison-Wesley, 1998.

[10] Debra J. Richardson, "*TAOS: Testing with Analysis and Oracle Support*", International Symposium on Software Testing and Analysis, page 138-153, March 1994.

[11] Nancy S. Eickelmann and Debra J. Richardson, "*An Evaluation of Software Test Environment Architectures*", International Conference on

Software Engineering, page 353-364, March 1996.

[12] Chia-Lin Hsu, "*A Web Database Application Model for Software Maintenance*", National Chiao-Tung Univertisy, Master Thesie, 1998.

[13] Ji-Tzay Yang, Jiun-Long Huang, and Feng-JianWang, "An Object-Oriented Architecture Supporting Web Application Testing," accepted by *J. Info.Sci.& Eng.*