Available online at www.sciencedirect.com
SCIENCE DIRECT
Information and Software Technology 48 (2006) 31–42
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Testing web applications

Nashat Mansour*, Manal Houri

Computer Science and Mathematics Division, Lebanese American University, Mme Curie Street, Beirut 11208, Lebanon

Received 10 September 2004; revised 17 February 2005; accepted 19 February 2005
Available online 17 May 2005

Abstract

Traditional testing techniques are not adequate for web-based applications, since they miss their additional features such as their multi-tier nature, hyperlink-based structure, and event-driven feature. Limited work has been done on testing web applications. In this paper, we propose new techniques for white box testing of web applications developed in the .NET environment with emphasis on their event-driven feature. We extend recent work on modeling of web applications by enhancing previous dependence graphs and proposing an event-based dependence graph model. We apply data flow testing techniques to these dependence graphs and propose an event flow testing technique. Also, we present a few coverage testing approaches for web applications. Further, we propose mutation testing operators for evaluating the adequacy of web application tests.

Keywords: Coverage testing; Data flow testing; Event flow testing; Dependence graphs; Mutation testing; Web applications

1. Introduction

The basic structure of web applications consists of three tiers: the client, the server and the data store. Web applications started simple and static and consisted mostly of HyperText Markup Language (HTML) pages. Then, the integration of HTML and other scripting languages has provided not only sophisticated web applications but also new issues that have to be addressed. The recent Microsoft's .NET platform has lowered the barriers to web software development [1]. Due to its widespread use, we focus on. NET web applications in this work; nevertheless, most of the techniques presented apply to web applications developed in various environments.

Active Server Pages (ASP.NET) supports event-driven programming. That is, objects on a web page can expose events that can be processed by ASP.NET code. ASP.NET also provides a language-neutral execution framework for web applications [1]. It includes a range of useful classes and namespaces. Namespaces are used as an organizational system, i.e. a way to present program components that are exposed to other programs and applications. ASP.NET provides several server controls that simplify the task of creating web pages. Server controls are tags that are understood by the server. These server controls encapsulate common tasks that range from displaying calendars and tables to validating user input. Traditional web applications contain a mix of HTML and scripts, making the code difficult to read, test, and maintain. ASP.NET alleviates this problem by promoting the separation of code and content using the code-behind feature. The user interface and the user interface programming logic need not be necessarily written in a single page. Thus, a web page consists mainly of a 'presentation file', that contains the HTML tags and the 'code-behind' class that contains methods and data items required for the class to perform specific actions. ASP.NET offers flexibility in usage of code-behind languages and allows a range of programming languages.

Testing is the process of revealing errors that is used to give confidence that the implementation of a program meets its specifications. Testing techniques are usually classified as black-box and white-box. Black-box methods are specification based such as equivalence partitioning, boundary value analysis, random testing and functional analysis-based testing [2,3]. White-box testing methods are code based such as statement testing, branch testing, path testing, predicate testing, dataflow testing, mutation testing

* Corresponding author. Tel.: +961 3 379647; fax: +961 1 867098.
E-mail addresses: nmansour@lau.edu.lb (N. Mansour), mhouri@idmnet.lb (M. Houri).

0950-5849/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2005.02.007

and domain testing [2,4–6]. In particular, mutation testing is a technique for evaluating the adequacy of test cases [7].

Web applications are complex, ever evolving, and rapidly updated software systems. Their testing is both challenging and critical. It is challenging because traditional testing methods and tools are not sufficient for web-based applications, since they do not address their distinctive features. Examples of the new features of web applications are: extensive use of events, rich graphical user interface, and incorporation of server side scripting. Testing web-based applications is critical because failure may be very costly. A failure in Amazon.com in 1998 put the site offline for several hours, with an estimated cost of $400,000 [8].

Research on web-based applications testing has been fairly limited. Some work has been recently published. Ricca and Tonnella [9] suggest a UML model of web applications and propose that all paths that satisfy selected criteria be tested. Ricca and Tonnella [10,11] investigate web application slicing and data flow testing of web applications; their work is further referred to in the body of this paper. Di Lucca [12] employs an object-oriented model of a web application and proposes to test single units of a web application as well as integration testing. Gabrys and Dick [13] suggest that testing web applications should be driven by factors that are most associated with achieving business objectives. Wu and Offutt [8] define a generic analysis model that characterizes typical behavior of web-based applications independently of different technologies. Elbaum et al. [14] explore the notion that user session data gathered as users operate web applications can be employed in the testing of those applications. Jia and Liu [15] propose an approach for testing web applications using formal specifications. In [16], data flow information of the web application using flow graphs is captured; test cases devised for these flow graphs are based on intra-object, inter-object, and inter-client perspectives.

In this paper, we present new white box techniques for testing web applications developed in the .NET environment. These techniques emphasize the distinct features of web-based programs, including their multi-tier nature, extensive use of events, and hyperlinked structure. First, we extend previous work on modeling web applications by enhancing previous dependence graphs and proposing an event-based dependence graph model. Second, we apply data flow testing methods to the dependence graphs and propose an event flow testing technique. Third, we present a few coverage testing approaches. Fourth, we propose mutation testing operators for evaluating the adequacy of web application tests. These techniques are illustrated with examples based on a .NET web application. Our approach to testing web applications is based on a fault model that characterizes plausible faults and classifies them into four types. The first type consists of navigation faults that are related to traversing the network of web pages presented by an application. Examples of such faults are: retrieving the wrong page or element upon executing a link; inconsistent sequence of retrieved pages vis-a-vis the navigation design. The second type of faults consists of interface faults that are related to interaction with users. Examples of these faults are: faults in the way graphical user interface elements implement the semantics of navigation or content display; missing or invalid data or wrong type (e.g. 'HIDDEN') for INPUT elements; graphical user interface elements (buttons, check boxes, etc.…) not enabled/disabled correctly. The third type of faults consists of integration faults that are related to the integration of collaborating web pages, of presentation and code-behind, and of code and databases. Examples of these faults are: collaboration/inheritance faults between presentation and code-behind parts; improper conformance of behavior with use-case scenarios and semantics; incorrect updating of databases in response to service requests (e.g. add, delete); wrong response/message of server-side code-behind parts in response to requests initiated at the presentation front end. The fourth type of faults consists of traditional faults such as faults in script functions and incorrect computations returned by processing methods of the application. The fourth type of faults is addressed using traditional testing techniques and is not pursued in this paper.

This paper is organized as follows. Section 2 gives dependence graph models for .NET web applications. Sections 3 and 4 present data flow testing, event flow testing, and coverage-based testing. Section 5 introduces mutation testing operators. Section 6 concludes the paper.

2. Dependence graph modeling

In this section, we present dependence graph models of.NET web applications. To illustrate our approach, we use a web application, 'To Do List', for a simple personal agenda shown in Fig. 1. This application contains only the essential elements of an agenda and was taken from [17]. It consists of two presentation ASP front ends (ToDoList.aspx, and EditItem.aspx) and two C# code-behind classes (todolist.aspx.cs and EditItem.aspx.cs) which are listed in the Appendix. As depicted in Fig. 1, the user views his/her unfinished tasks by priority order. Once finished with a task, the user can close it by clicking 'Done'. This removes the task from the open items and places it in the closed items. A user can also do other things such as 'Edit a task', 'Delete a task', 'Add item to agenda', 'Review item', etc.….

Some work has been reported on slicing web applications in [10,11] based on control, data, and call dependences. According to their work, a control/contain dependence holds between two statements if one defines a scope that directly contains the other; in the dependence graph, it is represented by a straight directed edge pointing to the dependent (contained) statement. Data dependence holds between two statements if one defines the value of a variable that is used by the other; in the dependence graph, it is represented by a curved directed edge. Fig. 2 illustrates control and data dependences. Also, call dependence holds