Web Testing for Reliability Improvement

JEFF TIAN[*] AND LI MA

Southern Methodist University,
Dallas, Texas, USA
tian@engr.smu.edu

Abstract
In this chapter, we characterize problems for web applications, examine existing testing techniques that are potentially applicable to the web environment, and introduce a strategy for web testing aimed at improving web software reliability by reducing web problems closely identified with web source contents and navigations. Using information about web accesses and related failures extracted from existing web server logs, we build testing models that focus on the high-usage, high-leverage subsets of web pages for effective problem detection and reliability improvement. Related data are also used to evaluate web site operational reliability in providing the requested pages as well as the potential for reliability growth under effective testing. Case studies applying this approach to the web sites www.seas.smu.edu and www.kde.org are included to demonstrate its viability and effectiveness. We also outline extensions to our approach to address testing, defect analysis, and reliability improvement issues for the constantly evolving web as a whole by analyzing the dynamic web contents and other information sources not covered in our current case studies.

[*]For correspondence, contact Dr. Jeff Tian, Computer Science & Engineering Dept., Southern Methodist University, Dallas, TX 75275, USA. Phone: +1-214-768-2861; fax: +1-214-768-3085.

1. Introduction

Web-based applications provide cross-platform universal access to web resources for the massive user population. With the prevalence of the world wide web (WWW) and people's reliance on it, testing and quality assurance for the web is becoming increasingly important. To help us test web-based applications and improve their reliability, we would like to adapt existing techniques that have been used effectively to assure and improve quality and reliability for traditional software systems [16,38,48]. As a prerequisite to successful adaptation, we must have a good understanding of the differences between the web environment and the traditional software systems, as well as a good understanding of the existing techniques, their applicability to different domains, and their effectiveness in dealing with different problems.

Quality in software is generally associated with good user experiences commonly characterized by the absence of observable problems and satisfaction of user expectations, which can also be intimately related to some internal characteristics of the software product and its development process [23,38,48]. A quantitative measure of quality meaningful to both the users and the developers is product *reliability*, which is defined as the probability of failure-free operations for a specific time period or input set under a specific environment [27,34]. Some testing and quality assurance techniques work directly to assure product reliability by detecting and correcting problems that are likely to be experienced by target customers and users [30,33], while others work indirectly to ensure some internal integrity for the product under development or in operation [5,35].

In subsequent sections, we review basic concepts and testing techniques, examine the characteristics of the web, discuss the use of existing models and techniques for web testing, and introduce an integrated web-testing strategy that ensures web reliability. In particular, Section 2 defines some basic terms related to defect and reliability to ensure a consistent interpretation of quality, and surveys major testing techniques and reliability models; Section 3 analyzes the web environment and its impact on web testing and quality assurance, particularly its challenges and opportunities, and focuses our attention on web problems closely identified with web source contents and navigations; Section 4 examines strategies for testing individual web elements and basic web navigation; Section 5 presents an integrated strategy for large-scale web testing that uses statistical testing models to selectively test important and frequently used web pages and to guide in-depth testing using traditional testing techniques; Section 6 uses defect and usage data from web logs to assess web site operational reliability and the potential for reliability growth under effective statistical testing, which also provide an external validation for our integrated testing strategy; and finally, we present our summary, conclusions, and future work in Section 7.

2. Basic Concepts and Techniques

As noticed above, various terms related to problems or defects are commonly used in discussing software quality and reliability. Several standard definitions [19] related to these terms include:

- *Failure*: The inability of a system or component to perform its required functions within specified performance requirements. It is an observable behavioral deviation from the user requirement or product specification.
- *Fault*: An incorrect step, process, or data definition in a computer program, which can cause certain failures.
- *Error*: A human action that produces an incorrect result, such as the injection of a fault into the software system.

Failures, faults, and errors are collectively referred to as *defects*. Testing plays a central role in assuring product quality by running the software, observing its behavior, detecting certain behavior deviations as failures, and helping us locate and fix the underlying faults that caused the observed failures.

Depending on whether external functions or internal implementation details are tested, we have two generic types of testing: *Functional* or *black-box* testing focuses on the external behavior of a software system, while viewing the object to be tested as a black-box that prevents us from seeing the contents inside [18]. *Structural* or *white-box* testing focuses on the internal implementation, while viewing the object to be tested as a white-box that allows us to see the contents inside [5].

Most of the traditional testing techniques and testing sub-phases use some coverage information as the stopping criteria [6,35], with the implicit assumption that higher coverage means higher quality or lower levels of defects. On the other hand, product reliability goals can be used as a more objective criterion to stop testing. The use of this criterion requires the testing to be performed under an environment that resembles actual usage by target customers so that realistic reliability assessment can be obtained, resulting in the so-called usage-based statistical testing [30,34].

The steps in most systematic testing include [7,48]:

(1) information gathering, with the internal implementations or external functions as the main information sources giving us white-box testing and black-box testing, respectively;
(2) test model construction, with models based on checklists, partitions, equivalence classes, and various forms of finite-state machines (FSMs) such as control flow graphs and data dependency graphs;
(3) test case definition and sensitization, which determines the input to realize a specific test;
(4) test execution and related problem identification; and
(5) analysis and followup, including making decisions such as what to test next or when to stop.

For usage-based statistical testing that play a critical role to ensure product reliability, the following particular steps are involved:

- The information related to usage scenarios, patterns, and related usage frequencies by target customers and users is collected.
- The above information is analyzed and organized into some models—what we call operational profiles (OPs)—for use in testing.
- Testing is performed in accordance with the OPs.
- Testing results is analyzed to assess product reliability, to provide feedback, and to support follow-up actions.

Both the failure information and the related workload measurements provide us with data input to various software reliability models that help us evaluate the current reliability and reliability change over time [27,34,44]. Two basic types of software reliability models are: input domain reliability models (IDRMs) and time domain software reliability growth models (SRGMs). IDRMs can provide a snapshot of the current product reliability. For example, if a total number of f failures are observed for n workload units, the estimated reliability R according to the Nelson model [36], one of the most widely used IDRMs, can be obtained as:

$$R = \frac{n-f}{n} = 1 - \frac{f}{n} = 1 - r.$$

Where r is the failure rate, which is also often used to characterize reliability. When usage time t_i is available for each workload unit i, the summary reliability measure, mean-time-between-failures (MTBF), can be calculated as:

$$\text{MTBF} = \frac{1}{f} \sum_i t_i.$$

If discovered defects are fixed over the observation period, the defect fixing effect on reliability (or reliability *growth* due to defect removal) can be analyzed by using various SRGMs [27,34]. For example, in the widely used Goel–Okumoto model [17], the failure arrival process is assumed to be a non-homogeneous Poisson process [22]. The expected cumulative failures, $m(t)$, over time t is given by the formula:

$$m(t) = N\left(1 - \mathrm{e}^{-bt}\right)$$

where the model constants N (total number of defects in the system) and b (model curvature) need to be estimated from the observation data.

3. Web Characteristics, Challenges, and Opportunities

Web applications possess various unique characteristics that affect the choices of appropriate techniques for web testing and reliability improvement. We next examine these characteristics in relation to existing software testing techniques and address general questions about web quality and reliability.

3.1 Characterizing Web Problems and Web Testing Needs

One of the fundamental differences between web-based applications and traditional software is the *document and information focus* for the former as compared to the computational focus for the latter. Although some computational capability has evolved in newer web applications, document and information search and retrieval still remain the dominant usage for most web users. Most of the documents and information sources are directly visible to the web users, as compared to the complicated background computation associated with traditional software systems. Consequently, the line that distinguishes black-box testing from white-box testing is blurred for the web environment.

A related difference between the development of web sites or web-based applications and that for traditional software systems is the increasingly faster pace and evolving nature of the former. Traditionally, large software systems take years to develop by a large group of professional developers, with the testing part typically consuming from a quarter to over half of the total development time and/or resources. In contrast, web sites and web-based applications can be set up and start running within a few weeks or even a few days by a small group of web developers. The dynamic web environment allows web site owners and contents providers to add, update, or change web contents, functions, and structures incrementally. All the testing sub-phases before product release for traditional software systems are condensed for the web environment, but post-release web site maintenance and related testing play an increasingly important role. This environment also provides opportunities for us to capture actual usage scenarios and patterns for effective web testing and reliability improvement for operational web site, as we describe in Section 5.

In addition, navigational facility is a central part of web-based applications, with the most commonly used HTML (hyper-text markup language) documents playing a central role in providing both information and navigational links. In this respect, web-based applications resemble many traditional menu-driven software products. The most commonly used testing technique for menu-driven software is the one based on finite-state machines (FSMs), where each menu is represented as a state in an FSM [6,48]. We next compare web-based applications to traditional menu-drive software products:

- Traditional menu-driven software still focuses on some computation; while web-based applications focus on information and documents.
- Traditional menu-driven software usually separates its navigation from its computation; while the two are tightly mingled for web-based applications.
- In traditional menu-driven software, there is usually a single top menu that serves as the entry point; while for web-based applications, potentially any web page can be the starting point. Similar differences exist for the end points or final states, with traditional menu-driven software having limited exits while web-based applications typically can end at any point when the user chooses to exit the web browser or stop browsing.
- There is a qualitative difference in the huge number of navigational pages even for moderately sized web sites and the limited number of menus for traditional menu-driven applications. We will revisit this key difference in Section 5 in connection with selective web testing.
- Web-based applications typically involve much more diverse support facilities than traditional menu-driven software. Web functionalities are typically distributed across multiple layers and subsystems, ranging from web browsers at the

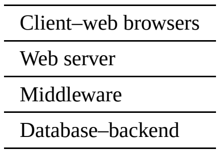

FIG. 1. Multi-layered web applications.

client side to web servers, middleware, and backend at the server side, as illustrated in Fig. 1. We need to make sure all different versions of web browsers, servers, and their underlying support infrastructure work well together to deliver the requested web contents.

Similar to general testing, testing for web applications focuses on reducing the chances for web failures by removing the underlying faults or defects. Therefore, we need to examine the common problems and associated concepts such as web failures, faults, and errors, before we can select, adapt, or develop appropriate testing techniques for this environment.

Based on the above characterization of the web environment, we can adapt some standard definitions of quality, reliability, defect, and related concepts to this new application environment. We define a web failure as the inability to correctly deliver information or documents required by web users [20]. This definition also conforms to the standard definition of failures as the behavioral deviations from user expectations [19] (correct delivery expected by web users). Based on this definition, we can consider the following failure sources:

- *Source or content failures*: Web failures can be caused by the information source itself at the server side.
- *Host or network failures*: Hardware or systems failures at the destination host or home host, as well as network failures, may lead to web failures. These failures are mostly linked to the web server layer and below in Fig. 1. However, such failures are not different from the regular system or network failures, and can be analyzed by existing techniques [43,54].
- *Browser failures*: Browser failures are linked to problems at the highest layer in Fig. 1 on the client side. These failures can be treated the same way as software product failures, thus existing techniques for software testing and quality assurance can be used [5,34,48].
- *User errors* can also cause problems, which can be addressed through user education, better usability design, etc. [3,11,42].

The last three groups of problems above can be addressed by the "global" web community using existing techniques, while web source or content failures are typically directly related to the services or functions that web-based applications are trying to provide. In addition, although usability is one of the primary concerns for novice web users, reliability is increasingly become a primary concern for sophisticated web users [55]. Therefore, we focus on web source failures and trying to ensure *reliability* of such web-based applications in this chapter.

With the above differentiation of different sources for web problems and the focus of source contents problems, we need to test the web (1) to ensure that individual components behave as expected, and (2) to ensure that the overall navigational facility works as expected. We will first describe testing models that attempt to "cover" those basic elements and navigation patterns in Section 4. However, such coverage-based testing would soon run into difficulties for web sites with numerous functions or diverse usage patterns, leading us to usage-based web testing in Section 5.

3.2 Information Sources for Testing and Analyses: New Opportunities

Information sources for testing models can generally be divided into two broad categories: (1) internal components and structures and (2) external functions and related usages. For the web environment, the easy access to source files (primarily HTML documents) is a significant difference from traditional software systems where the users typically do not have access to program source code. This difference would blur the distinction between black-box testing and white-box testing, and allow independent testers and users to perform white-box testing for different web components in addition to black-box testing.

Although external functions for web-based applications and web sites are rarely formally specified, we do have abundant information about web usages available, typically recorded in various server log files. In fact, monitoring web usage and keeping various logs are necessary to keep a web site operational. Therefore, we would only incur minimal additional cost to use the information recorded in these logs for testing and reliability analyses.

Two types of log files are commonly used by web servers: Individual web accesses, or hits, are recorded in *access logs*, and related problems are recorded in *error logs*. A "hit" is registered in the access log if a file corresponding to an HTML page, a document, or other web content is explicitly requested, or if some embedded content, such as graphics or a Java class within an HTML page, is implicitly requested or activated. Some sample entries from the access log for the www.seas.smu.edu web site using Apache Web Server [4] is given in Fig. 2. Specific information in this access log includes:

```
129.119.4.17 - - [16/Aug/1999:00:00:11 -0500] "GET
/img/XredSeal.gif HTTP/1.1" 301 328 "http://www.seas.smu.edu/"
"Mozilla/4.0 (compatible; MSIE 4.01; Windows NT)"

129.119.4.17 - - [16/Aug/1999:00:00:11 -0500] "GET
/img/ecom.gif HTTP/1.1" 304 - "http://www.seas.smu.edu/"
"Mozilla/4.0 (compatible; MSIE 4.01; Windows NT)".
```

FIG. 2. Sample entries in an access log.

- The reverse-DNS hostname of the machine making the request. If the machine has no reverse-DNS hostname mapped to the IP number, or if the reverse-DNS lookup is disabled, this will just be the IP number.
- The user name used in any authentication information supplied with the request.
- If “identd” checking is turned on, the user name as returned by the remote host.
- Date and time that the transfer took place, including offset from Greenwich Mean Time.
- The complete first line of the HTTP request, in quotes.
- The HTTP response code.
- Total number of bytes transferred.
- The referrer, or the source page that lead to the current access.
- The agent, or the information that the client browser reports about itself.

If the value for any of these data fields is not available, a “−” will be put in its place. Most of the above information is available from most web logs despite minor variations in web log configurations and information contents across different web servers.

Error logs typically include details about the problems encountered. The format is simple: a time-stamp followed by the error or warning message, such as in Fig. 3.

```
[Mon Aug 16 13:17:24 1999] [error] [client 207.136.6.6]
File does not exist: /users/seasadm/webmastr/htdocs/
library/images/gifs/homepage/yellowgradlayers.gif

[Mon Aug 16 13:17:37 1999] [info] [client 199.100.49.104]
Fixed spelling: /img/XredSeal.gif to /img/xredSeal.gif
from http://www.seas.smu.edu/
```

FIG. 3. Sample entries in an error log.

Analyzing information stored in such logs can help us capture the overall web usages for testing, as described in Section 5. Various measurements can be derived to characterize web site workload at different levels of granularity and from different perspectives. These workload measurements are used together with failure information in Section 6 to evaluate web site operational reliability and the potential for reliability improvement. Web logs also give us a starting point to characterize common problems for web-based applications, as we discuss below.

3.3 Preliminary Analysis of Common Problems for Two Web Sites

We next analyze common problems for www.seas.smu.edu, the official web site of the School of Engineering and Applied Science at Southern Methodist University (SMU/SEAS). This web site utilizes Apache Web Server [4], a popular choice among many web hosts, and shares many common characteristics of web sites for educational institutions. These features make our results and observations here and in the rest of this chapter meaningful to many application environments. Server log data covering 26 consecutive days in 1999 were used.

The use of this web site continues the trend of most previous studies that overwhelmingly focus on academic sites [39], which may not be a good representative for many other web sites. For example, most of the SMU/SEAS web pages are static ones, consisting primarily of the HTML documents and embedded graphics [25], and the web site operates under fairly light traffic. In e-commerce and other applications, workload types may be more diverse, with dynamic pages and context-sensitive contents play a much more important role, and traffic volume can be significantly larger [39]. Therefore, we obtained recent (2003) web logs from the open source KDE project web site www.kde.org to cross-validate our results [52]. The overall user population and traffic volume are significantly larger, and changes are continuously committed to the web site in order to provide the developers and users with the most up-to-date information. These characteristics make it a good choice for our validation study. On the other hand, this web site also uses Apache Web Server [4], which makes our data extraction and analysis easy due to the same data format used.

Common web problems or error types are listed in Table I. Notice that most of these errors conform closely to the source content failures we defined above, making them suitable for our web problem characterization and reliability evaluation. Table I also gives the summary of different types of errors for SMU/SEAS. The most dominant error types are type A (“permission denied”) and type E (“file does not exist”), which together account for 99.9% of the recorded errors.

TABLE I
GENERIC ERROR TYPES AND THE RECORDED ERRORS BY TYPE FOR SMU/SEAS

Type	Description	# of errors
A	permission denied	2079
B	no such file or directory	14
C	stale NFS file handle	4
D	client denied by server configuration	2
E	file does not exist	28,631
F	invalid method in request	0
G	invalid URL in request connection	1
H	mod_mime_magic	1
I	request failed	1
J	script not found or unable to start	27
K	connection reset by peer	0
All types		30,760

Type A errors, accounting for 6.8% of the total recorded errors, involve improper access authorization or problems with the authentication process. These errors are more closely related to security problems instead of reliability problems we focus on in this chapter. Further analyses of them may involve the complicated authentication process. In addition, type A errors also account for much less of a share of the total recorded errors than type E errors. Therefore, we decided not to focus on these errors in our study.

Type E errors usually represent bad links. They are by far the most common type of problems in web usage, accounting for 93.1% of the total recorded errors. This is in agreement with survey results from 1994–1998 by the Graphics, Visualization, and Usability Center of Georgia Institute of Technology (see http://www.gvu.gatech.edu/user_surveys). The surveys found that broken link is the problem most frequently cited by web users, next only to network speed problem. Therefore, type E error is the most observed web content problem for the general population of web users. Further analysis can be performed to examine the trend of these failures, to guide web testing, and to provide an objective assessment of the web software reliability.

For the KDE web site, only access logs are used but not the error logs. However, from the HTTP response code, we can extract the general error information. For example, type E (missing file) errors in the error logs are equivalent to access log entries with a response code 404. The access logs for the KDE web site for the 31 days recorded more than 14 million hits, of which 793,665 resulted in errors. 785,211 hits resulted in response code 404 (file-not-found), which accounted for 98.9% of all the errors. The next most reported error type was of response code 408, or “request timed out," which accounted for 6225 or 0.78% of all the errors. This dominant share of 404 errors, which is equivalent to type E errors, justifies our focus on this type of errors in our study.

We performed a preliminary analysis of the originators of these bad links [28] and discovered that the majority of them are from internal links, including mostly URLs embedded in some web pages and sometimes from pages used as start-ups at the same web site. Only a small percentage of these errors are from other web sites (4.3%), web robots (4.4%), or other external sources, which are beyond the control of the local site content providers, administrators, or maintainers. Therefore, the identification and correction of these internal problems represent realistic opportunities for improved web software reliability based on local actions such as testing and other quality assurance activities. In addition, we also need to test for problems beyond what is reported in web logs, as we describe next.

4. Testing Individual Web Elements and Navigations

For the web functionalities distributed across multiple layers and subsystems, we need to make sure that all the components work well individually as well as together. We next examine the techniques to test these individual web components and the basic web navigations.

4.1 Web Components, Aspects, and Testing

Web components [29] that need to be tested for conformance to some standards or user expectations include the following:

- *HTML document*, still the most common form for documents on the web.
- *Java, JavaScript, and ActiveX* commonly used to support platform independent executions.
- *Cgi-Bin Scripts* used to pass data or perform some other activities.
- *Database*, a major part of the backend.
- *Multi-media components* used to present and process multi-media information.

All these components need to be tested. Fortunately, many existing testing techniques and tools can be used to perform such testing. In fact, most existing work on web testing focuses on *functionality testing* to test web components to ensure that the web site performs its intended functions as expected [9,15]. This type of testing usually involves analyzing given web components and checking their conformance to relevant standards and external specifications. Specific types of functionality testing include:

- *HTML syntax and/or style checking*: HTML validators, such as Weblint (www.weblint.org), W3C Validator (validator.w3.org), and CSSCheck (www.htmlhelp.com/tools/csscheck/), can parse HTML files and check their conformance to relevant language specifications and document standards.
- *Link checking* can be performed to check the entire site for broken links, with the help of tools like Net Mechanic (www.netmechanic.com). This is similar to link coverage testing in FSM-based testing described later in this section for testing web navigations, but without formally constructing an FSM.
- *Form testing* checks input types and variable names in various forms, with the help of tools such as Doctor HTML (www2.imagiware.com/RxHTML). This can be considered as rudimentary input domain testing based on a simple checklist [48].
- *Verification of end-to-end transactions*, which is similar to testing complete execution paths in control flow testing or transaction processing in transaction flow testing [6].
- *Java and other component testing*: Java applets, which work on the clients side, or other Java applications, which work on the server side, need to be tested, similar to traditional software testing. Similarly, other programming languages used for various web-based facilities, such as C/C++, Visual Basic, etc., can also be checked for syntax, decisions, etc.

Various tools or tool suites are also available to support a combination of various functionality and regression testing, including SilkTest (www.segue.com), Astra QuickTest (www.mercury.com), eTester (www.rswsoftware.com), etc. Besides the basic web elements that need to be tested above, some specific web aspects that cut through several web elements also need to be tested, including:

- *Load testing* is a subset of stress (or performance) testing. It verifies that a web site can handle a large number of concurrent users while maintaining acceptable response time.
- The focus of *usability testing* is the ease-of-use issues of different web designs, overall layout, and navigations [3,11,29], which is different from our reliability focus in this chapter. Such testing relies heavily on subjective preferences of selected users.
- *Browser rendering* problems may affects the delivery as well as presentation of web contents. For example, HTML files that look good on one browser may look bad on another. We need to make sure that the web site functions appropri-

ately with different browser versions. However, the browser checking is done manually to assess the “look & feel” of the GUI, etc., similar to usability testing discussed above.

- Other specialized testing include accessibility for blind and disabled people, runtime error detection, web security testing, etc. [29,37].

4.2 Modeling and Testing WEB Navigations as Finite-State Machines (FSMs)

As mentioned in Section 3, web navigations have many similarities with menu selections in traditional menu-driven software systems, and the most commonly used testing technique for such systems is based on finite-state machines (FSMs) [5,48]. From the web users’ point of view, each web-based application or function consists of various components, stages, or steps, visible to the web users, and typically initiated by them, making FSM-based testing an appropriate choice. We next consider the basic elements of FSMs and map them to web-based applications:

- Each web page corresponds to a state in an FSM. When we start a web browser, the default starting page or our customized starting page will be loaded, which corresponds to the initial state. Therefore, potentially any page can be the initial state. Similarly, we can stop anytime by exiting the web browser, or implicitly by no longer requesting pages. This last page visited, which can be potentially any page, is then the final state.
- State transitions correspond to web navigations following hypertext links embedded in HTML documents and other web contents.
- The input and output associated with such navigations are fairly simple and straightforward: The input is the clicking of the embedded link shown as highlighted content. The corresponding output is the loading of the requested page or content with accompanying messages indicating the HTML status, error or other messages, etc.

One special case in the state transition modeling above is that a user may choose to follow a previous saved link (bookmarked favorites) or to directly type a URL (universal resource locator, the address of a specific page). The use of these external navigation tools makes state transitions more unpredictable. However, there are also two factors worth noting in modeling web navigations as state transitions in FSMs:

- From the point of view of Internet- and web-based service providers, it is more important to ensure that the “official” embedded links on the providers’ web site are correct than to ensure that the users’ bookmarks or typed URLs are up-to-date or correct.

- There is empirical evidence that the vast majority of web navigations are following embedded hypertext links instead of using bookmarked or typed URLs. For example, for the www.seas.smu.edu web site, 75.84% of the navigations are originated from embedded links within the same web site, only 12.42% are user originated, and the rest from other sources [28].

Consequently, we choose to focus on the embedded navigation links and capture them in FSMs for web testing.

Once these FSMs are constructed, they can be used in testing. FSM-based testing typically attempts to achieve the following coverage goals [5,48]:

- *State or node coverage*: We need to make sure that each state can be reached and visited by some test cases. This is essentially a state or node traversal problem in graph theory [13,24], and test cases can be derived accordingly. In fact, web robots used by various Internet search engines or index services commonly "crawl" the web by systematically following the embedded hypertext links to create indexes or databases of the overall web contents, much like the state traversal for FSMs.
- *Transition or link coverage*: We need to make sure that each link or state transition is covered by some test cases. Although this problem can also be treated as link traversal in graph theory, the above state coverage testing already helped us reach each reachable state. It would be more economical to combine the visit to these states with the possible input values to cover all the links or transitions originated from this current state, which would also help us detect missing links (some input not associated with any transitions) [48]. Existing link-checkers can also be used to help us perform link coverage testing by checking bad links ("`file-not-found`"), which represent extra links in FSMs or missing states (pages).

In trying to reach a specific state, each test case is essentially a series of input values that enables us to make the transitions from an initial state to some target state, possibly through multiple hops by way of some intermediate states. The key in this sensitization is to remember that in FSM-modeled systems, input and output are associated with individual transitions instead of as an indistinguishable lump of initial input for many other systems. Consequently, the input sequencing is as important as the correct input values.

One useful capability for test execution is the ability to save some "current state" that can be restored. This would significantly shorten the series of state transitions needed to reach a target state, which may be important because in some systems these transitions may take a long time. This capability is especially useful for link coverage testing starting from a specific state: If we can start from this saved state, we can go directly into link coverage testing without waiting for the state transitions to reach this state from an initial state. Fortunately for web testing, most web pages can be saved or "bookmarked" to enable us to perform such testing easily. For a subset of dynamic and embedded pages, a more complicated navigation sequence will probably be needed.

The result checking is easy and straightforward, since the output for each transition is also specified in FSMs in addition to the next state. The key to this result checking is to make sure that both the next state and the output are checked.

4.3 Limitations and Motivation for Usage-Based Web Testing

We characterized web-based applications in Section 3 by their information/document focus, integration between information and navigation, and multi-layered support infrastructure to derive checklist- and FSM-based testing above for web-based applications. Additional characteristics of web-based applications include:

- *Massive user population*: Virtually anyone from anywhere with an Internet access can be a user of a given web-site. Although some traditional software systems, such as operating systems, also serve a massive user population, the systems are usually accessed locally, thus scattering the user population into sub-groups of limited size.
- *Diverse usage environments*: Web users employ different hardware equipments, network connections, operating systems, middleware and web server support, and web browsers, as compared to pre-specified platforms for most traditional software.

Any reliability problem of the web-based applications will be magnified by the massive user population, requiring us to address reliability problems effectively and directly. The diverse usage environment requires thorough testing to be performed for a huge number of situations. However, traditional coverage-based testing adapted for web testing in this section cannot be used directly to ensure reliability for web-based applications, and the combinatorial explosion resulted from the above diverse environments would make "coverage" an unattainable goal, as discussed below.

There is one obvious drawback to web testing using techniques covered in this section: The number of web pages for even a moderate-sized web site can be thousands or much more. Consequently, there would be significant numbers of unorganized individual testing activities when we attempt to test the individual items or aspects, which would overwhelm testing resources. If FSMs are used, the large number of states makes any detailed testing impractical, even with some automated support, because FSM-based testing can generally handle up to a few dozen states at most [5,48].

Hierarchical FSMs can be used to alleviate the problem associated with the large numbers of states and transitions by limiting transactions across boundaries of different FSMs in the hierarchy: In lower-level models, we generally assume a common source and common sink as its interface with higher-level models. However, the interactions may well be cutting through hierarchy boundaries in real systems. For large web sites, the complete coverage of all these hierarchical FSMs would still be impractical.

Another alternative to deal with state explosion problem of FSMs is to use selective testing based on Markov operational profiles (Markov OPs) [30,56] by focusing on highly used states and transitions while omitting infrequently used ones or grouping them together. The combination of these hierarchical FSMs and Markov OPs led us to unified Markov models (UMMs) [20,49] described in Section 5. On the other hand, loosely related collections of web pages can be more appropriately represented and tested by using some simpler usage models based on a flat list of operations and associated probabilities, such as Musa's operational profiles (OPs) [33]. The introduction of statistical testing strategies based on such OPs or UMMs is not to replace traditional testing techniques, but to use them selectively on important or frequently used functions or components.

On the other hand, as a general rule, usage and problem distribution among different software components is highly uneven [8], which is also demonstrated to be true among different web contents [25]. Consequently, some kind of selective testing is needed to focus on highly-used and problematic areas to ensure maximal web site reliability improvement, such as through usage-based statistical testing we discuss in the next section. These techniques can help us prioritize testing effort based on usage scenarios and frequencies for individual web resources and navigation patterns to ensure the reliability of web-based applications.

5. A Hierarchical Approach for Large-Scale Web Testing

We next describe an integrated strategy that combines several existing testing techniques in a hierarchical framework for web testing [48,50]. We first outline the overall framework and then describe its individual tiers in detail.

5.1 Overall Framework: A Three-Tiered Web Testing Strategy

Once we have decided to use usage-based statistical web testing, the immediate question is then the choice between two commonly used usage models or operational profiles (OPs): Musa's flat OP [33] and Markov OP [30,56] or its variation, unified Markov models (UMMs) [20,49]. Musa OP's key advantage is its simplicity, allowing us to focus on frequently used individual pages or web components. On the other hand, Markov models such as UMMs are generally more appropriate for selective testing of web navigations. Therefore, we have selected UMMs as the central testing models to work under the guidance of high-level Musa OPs in our overall web testing framework, to form a three-tiered strategy [50]:

1. The high-level operational profile (OP) enumerates major functions to be supported by web-based applications and their usage frequencies by target customers. This list-like flat OP will be augmented with additional information and supported by lower-level models based on unified Markov models (UMMs). The additional information includes grouping of related functions and mapping of major external functions to primary web sources or components.
2. For each of the high-level function groups, a UMM can be constructed to thoroughly test related operation sequences and related components. UMMs capture desired behavior, usage, and criticality information for web-based applications, and can be used to generate test cases to exhaustively cover high-level operation sequences and selectively cover important low-level implementations. The testing results can be analyzed to identify system bottlenecks for focused remedial actions, and to assess and improve system performance and reliability.
3. Critical parts identified by UMMs can be thoroughly tested using lower-level models based on traditional testing techniques. Other quality assurance alternatives, such as inspection, static and dynamic analyses, formal verification, preventive actions, etc. [38,48], can also be used to satisfy user needs and expectations for these particular areas.

Therefore, existing techniques that attempt to "cover" certain web aspects or elements, particularly those we described in Section 4, can still be used, but under the guidance of our mid-level UMMs as well as our top-level Musa OPs. Our hierarchical strategy can also accommodate other quality assurance activities, mostly at the bottom-level, much like for coverage-based testing:

- Support for software inspection can be provided in two ways: (1) selective inspection of critical web components by relating frequently used services to specific web components; (2) scenario based inspection [40] guided by our UMMs where usage scenarios and frequencies can be used to select and construct inspection scenarios.
- Selective formal verification can be carried out similar to the way inspection is supported above. Specific formal verification models can also be associated

with highly critical parts of UMMs, and much of the UMM information can help us work with formal verification models, particularly those based on symbolic executions [31,57].

- The ability to identify reliability bottlenecks can also help the selective application of corrective and preventive actions, system analyses, damage containment, etc. [48].

To implement the above strategy, quantitative web usage information needs to be collected. Fortunately, the availability of existing web logs offers us the opportunity to collect usage information for usage model construction and for statistical web testing. The following reports can be easily produced from analyzing the web access logs kept at the web servers:

- *Top access report* (TAR) lists frequently accessed (individual) services or web pages together with their access counts.
- *Call pair report* (CPR) lists selected important call pairs (transition from one individual service to another) and the associated frequency.

TAR is important because many of the individual services can be viewed as standalone ones in web-based applications, and a complete session can often be broken down into these individual pieces. This report, when normalized by the total access count or session count, resembles the flat OP [33]. Each service unit in a TAR may correspond to multiple pages grouped together instead of a single page, such as in Fig. 2. Such results provide useful information to give us an overall picture of the usage frequencies for individual web service units, but not navigation patterns and associated occurrence frequencies.

CPR connects individual services and provides the basic information for us to extract state transition probabilities for our UMMs. For a web site with N pages or units, the potential number of entries in its global CPR is N^2, making it a huge table. Therefore, we generally group individual pages into larger units, similar to what we did above for TAR. Alternatively, we can restrict CPR to strong connections only, which is feasible because the $N \times N$ table is typically sparsely populated with only a few entries with high cross-reference frequency, as we will see empirically in Fig. 9 at the end of this section.

We can traverse through CPR for strong connections among TAR entries, which may also include additional connected individual services not represented in TAR because of their lower access frequencies or because they represent lower-level web units. A UMM can be constructed for each of these connected groups. In this way, we can construct our UMMs from TAR and CPR.

Notice that multiple OPs, particularly multiple UMMs in addition to TAR, our top-level OP, usually result for a single set of web-based applications using the above

Top Level:	Top Access Report (TAR) a flat list of frequently accessed services in ranking order (may be grouped by interconnection in customer usage scenarios)
Middle Level:	Unified Markov Models (UMMs) for groups of TAR entries linked by CPR (call-pair report) (may be expanded into lower-level UMMs or other models)
Bottom Level:	Detailed UMMs or other Models associated with frequently visited or critical nodes of UMMs (may correspond to testing models other than UMMs)

FIG. 4. Hierarchical implementation of an integrated web testing strategy.

approach. This implementation of our integrated strategy in a hierarchical form is discussed below:

- At the top level, TAR can be used directly as our flat OP for statistical usage-based testing.
- Entries in TAR can be grouped according to their connections via CPR, and a UMM can be constructed for each of these groups, forming our middle-level usage models, or our individual UMMs.
- The hierarchical nature of our UMMs will allow us to have lower-level UMMs as well as other lower-level testing models to thoroughly test selected functions or web components.

This hierarchical implementation of our integrated strategy is graphically depicted in Fig. 4. We focus on testing frequently used individual functions or services at the top level, testing common navigation patterns and usage sequences at the middle level, and covering selected areas at the bottom level. Specific low-level UMMs or other coverage-based testing models can be built to thoroughly test the related features or critical components in the higher-level flat OPs or UMMs. Coverage, criticality, and other information can also be easily used to generate test cases using lower-level models under our OPs.

5.2 Testing High-Level Component Usage with Musa Operational Profiles

According to Musa [33], an operational profile (OP) is a list of disjoint set of operations and their associated probabilities of occurrence. It is a quantitative characterization of the way a software system is or will be used. Table II gives an example OP for the web site, www.seas.smu.edu, listing the number of requests for differ-

TABLE II
USAGE FREQUENCIES AND PROBABILITIES BY FILE TYPES FOR SMU/SEAS

File type	Hits	% of total
.gif	438,536	57.47
.html	128,869	16.89
directory	87,067	11.41
.jpg	65,876	8.63
.pdf	10,784	1.41
.class	10,055	1.32
.ps	2737	0.36
.ppt	2510	0.33
.css	2008	0.26
.txt	1597	0.21
.doc	1567	0.21
.c	1254	0.16
.ico	849	0.11
Cumulative	753,709	98.78
Total	763,021	100

ent types of files by web users over 26 days and the related probabilities. This OP can also be viewed as a specialized TAR above, where individual web pages are grouped by file types to form individual service units. The adaptation and application of OP-based testing would ensure that frequently used web pages and components are adequately tested, which in turn, would have a great impact on web site reliability improvement.

The "operations" represented in the operational profiles are usually associated with multiple test cases or multiple runs. Therefore, we typically assume that each "operation" in an OP can be tested through multiple runs without repeating the exact execution under exactly the same environment. In a sense, each operation corresponds to an individual sub-domain in domain partitions, thus representing a whole equivalence class [48]. In this example, each item in the table, or each operation, represents a type of file requested by a web user, instead of individual web pages. Of course, we could represent each web page as an operation, but it would be at a much finer granularity. When the granularity is too fine, the statistical testing ideas may not be as applicable, because repeated testing may end up repeating a lot of the same test runs, which adds little to test effectiveness. In addition, such fine-granularity OPs would be too large to be practical. For example, the SMU/SEAS web site has more than 11,000 individual web pages, while the number of file types is limited to a hundred or so, including many variations of the same type, such as HTML files with extensions of ".HTML," ".html," ".htm," etc.

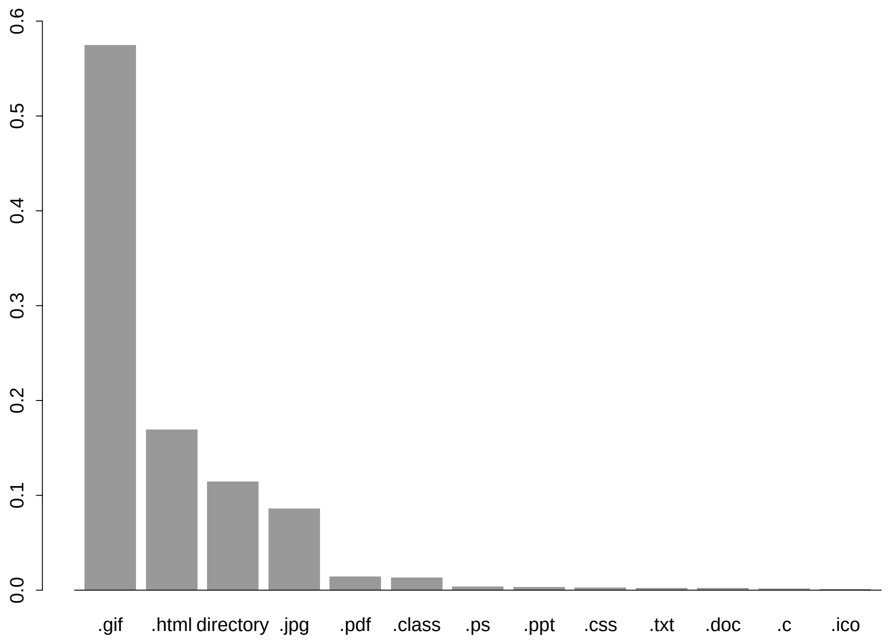

FIG. 5. An operational profile (OP) of requested file types for the SMU/SEAS web site.

There are also several other key points worth noting about Musa OPs, including:

- It is customary to sort the operations by descending probabilities of usage and present the results in that order. This sorted OP is often represented visually as a histogram, such as in Fig. 5, to make the results easy to interpret.
- It is common to have quite uneven distribution of usage probabilities, with a few frequently used ones account for most of the usage frequencies. In Table II, the top 13 out of a hundred or so file types account for more than 98% of the web hits for SMU/SEAS.
- Related to the uneven distribution of usage probabilities is the probability threshold for individual operations. The basis of statistical testing is to perform more testing for those operations that are used more by the customers. Therefore, if some operations have very low probability of usage, we could omit them in the OP. This probability threshold plays an important role in limiting the numbers of operations to represent in the OP, especially when there are a large number of possible operations.

There are three generic methods for information gathering and OP construction, in decreasing accuracy: actual *measurement* of usage at customer installations, *survey* of target customers, and usage estimation based on *expert opinions*. Fortunately, the availability of existing web logs offers us the opportunity to collect usage information for OP construction without incurring much additional cost. For new web sites or new web-based applications, similar information about the "intended" customer usage can be obtained by surveying potential customers or from experts.

Once an OP is constructed, it can be used to support statistical testing by some random sampling procedure to select test cases according to the probability distribution and execute them. Essentially, each operation in the OP corresponds to certain test cases specifically constructed or selected from existing ones to test a specific system operation.

The actual test runs are sampled from these test cases according to the probability of associated operations. The number of test runs for each operation in the OP is proportional to its probability. Under most circumstances, these test cases and associated runs can be prepared ahead of time, so that some test procedure can be employed to sequence the multiple test runs according to various criteria. In some cases, truly random sampling can be used, to dynamically select test cases to run next. However, such dynamic random sampling will slow down test execution and the system performance because of the overhead involved in managing the test case selection in addition to monitoring the test runs. Therefore, unless absolutely necessary, we should prepare the test cases and test procedures ahead of time to reduce the impact of testing overhead on normal system operations.

In addition to or in place of proportional sampling, progressive testing is often used with the help of OPs. For example, at the beginning of testing, higher probability threshold can be used to select a few very important or highly used operations for testing. As testing progresses, the threshold can be lowered to allow testing of less frequently used operations so that a wide variety of different operations can be covered. In a sense, the use of OPs can help us prioritize and organize our testing effort so that important or highly used areas are tested first, and other areas are progressively tested to ensure good coverage.

5.3 Unified Markov Models (UMMs) for Usage-Based Testing

As mentioned in Section 4, the primary limitation of FSM-based testing for the web is its inability to handle large number of states or individual web pages. We can augment FSMs with probabilistic usage information to focus our testing effort on usage scenarios and navigation sequences commonly used by target customers, while reduce or eliminate testing of less frequently used ones. The use of this approach would help us ensure and maximize product reliability from a customer's perspective, and at the same time make it scalable. Such augmented FSMs are our OPs, which typically form Markov chains, as we describe below.

A Markov chain can be viewed as an FSM where the probabilities associated with different state transitions satisfy the so-called *Markovian* or *memoryless* property: From the current state $X_n = i$ at time n or stage n, the probability of state transition to state $X_{n+1} = j$ for the next time period or stage $n+1$ is denoted as p_{ij}, which is independent of the history, i.e.,

$$\begin{aligned} &P\{X_{n+1} = j \mid X_n = i,\ X_{n-1} = s_{n-1},\ \ldots,\ X_0 = s_0\} \\ &\quad = P\{X_{n+1} = j \mid X_n = i\} = p_{ij} \quad \text{where } 0 \leqslant p_{ij} \leqslant 1 \text{ and } \sum_j p_{ij} = 1. \end{aligned}$$

In other words, the probability that the system will be in state j only depends on the current state i and the history-independent transition probability p_{ij}. Equivalently, the complete history is summarized in the current state, thus removing the need to look back into the past to determine the next transition to take. This property is call the memoryless or Markovian property in stochastic processes [22]. A simplified test of memoryless property has been proposed and successfully used to verify that web usages can be accurately modeled by Markov chains [26].

Figure 6 is a sample Markov chain, with probabilistic state transitions. For example, after state B the next state to follow is always C, as represented by the transition probability of $p(B, C) = 1$. While the states to follow C could be D, with probability $p(C, D) = 0.99$ for the normal case, or B, with probability $p(C, B) = 0.01$ for the rare occasion that MS is unable to receive paging channel. Notice that we omitted the input/output information in such Markov OPs to keep the illustration simple, with the understanding that the input/output information is available for us to sensitize testing.

Much of the previous work with statistical testing used individual Markov chains [30,56]. For large systems, a collection of Markov chains might be used for testing, organized into a hierarchical framework called unified Markov models (UMMs) [20, 49,50]. For example, the top-level Markov chain for call processing in a cellular communication network is represented in Fig. 6. However, various sub-operations may be associated with each individual state in the top-level Markov chain, and could be modeled by more detailed Markov chains, such as the one in Fig. 7 for expanded state E. Notice that in some of these Markov chains, the sum of probabilities for transitions out from a given state may be less than 1, because the external destinations (and sources) are omitted to keep the models simple. The implicit understanding in UMMs is that the missing probabilities go to external destinations.

The higher-level operations in a UMM can be expanded into lower-level UMMs for more thorough testing. Therefore, UMMs are more suitable for large systems

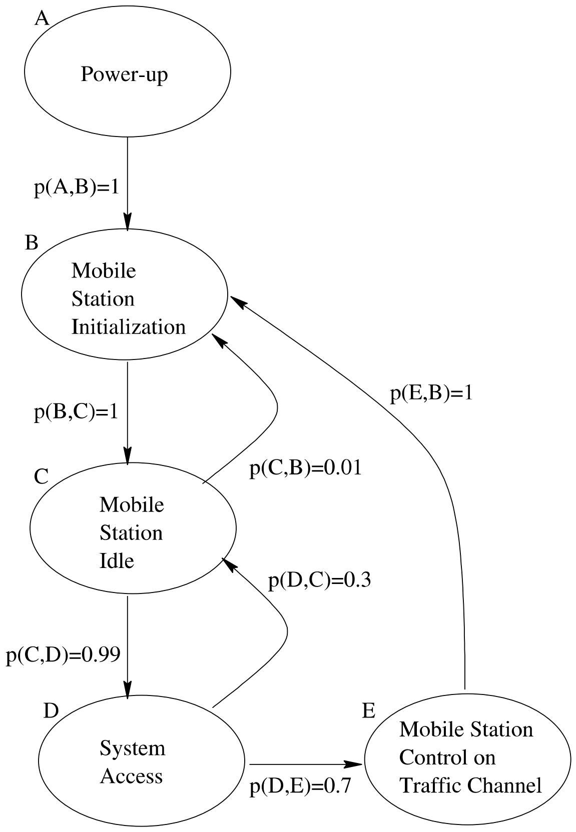

FIG. 6. An example Markov chain.

with complex operational scenarios and sequences than Musa's OPs above or deterministic FSMs described in Section 4. This hierarchical structure and the associated flexibility set this approach apart from earlier approaches to statistical testing using Markov chains.

Test cases can be generated by following the states and state transitions in UMMs to select individual operational units (states) and link them together (transitions) to form overall end-to-end operations. Possible test cases with probabilities above specific thresholds can be generated to cover frequently used operations. In practical applications, thresholds can be adjusted to perform progressive testing, similar to that used with Musa OPs we described earlier. Several thresholds have been initially proposed [2] and used in developing UMMs [20,49]:

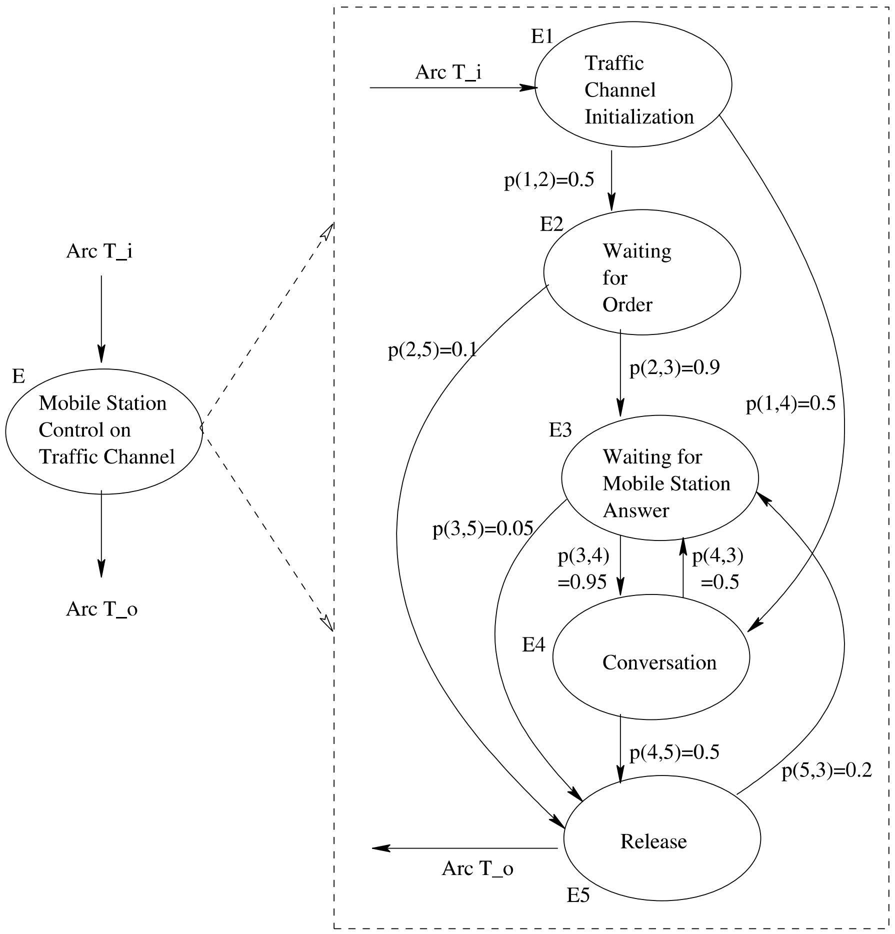

FIG. 7. Expanding state E of the top-level UMM in Fig. 6 into a lower-level UMM.

- *Overall probability threshold* for complete end-to-end operations to ensure that commonly used complete operation sequences by web users are adequately tested.
- *Stationary probability threshold* to ensure that frequently visited states are adequately tested.
- *Transition probability threshold* to ensure commonly used operation pairs, their interconnections and interfaces are adequately tested.

In our hierarchical strategy for web testing, information captured in our TAR and CPR is similar to the stationary probability and transition probability above. Consequently, we rely less on the last two thresholds above, and instead primarily use the overall probability threshold for our testing with UMMs. To use this threshold, the probability for possible test cases need be calculated and compared to this threshold. For example, the probability of the sequence ABCDEBCDC in Fig. 6 can be calculated as the products of its transitions, that is,

$$1 \times 1 \times 0.99 \times 0.7 \times 1 \times 1 \times 0.99 \times 0.3 = 0.205821.$$

If this is above the overall end-to-end probability threshold, this test case will be selected and executed.

Coverage, importance and other information or criteria may also be used to generate test cases. In a sense, we need to generate test case to reduce the risks involved in different usage scenarios and product components, and sometimes to identify such risks as well [14]. The direct risks for selective testing include missing important areas or not covering them adequately. These “important” areas can be characterized by various external or internal information. The coverage requirement can be handled similarly by adjusting the probabilities to ensure that all things we would like to cover stays above a certain threshold, or by adjusting our test case selection procedures. Similar adjustments as above can be used to ensure that some critical functions of low usage frequencies are also thoroughly tested.

The hierarchical structure of UMMs also gives us the flexibility to improve test efficiency by avoiding redundant executions once a subpart has been visited already. This is particularly true when there are numerous common sub-operations within or among different end-to-end operations. When revisiting certain states, exact repetition of the execution states that have been visited before is less likely to reveal new problems. The revisited part can be dynamically expanded to allow for different lower-level paths or states to be covered. For example, when state E is revisited in the high-level Markov chain in Fig. 6, it can be expanded by using the more detailed Markov chain in Fig. 7, and possibly execute different sub-paths there. In general, to avoid exact repetition, we could expand revisited states with operations of finer granularity, and more thoroughly test those frequently used parts.

5.4 Constructing and Using UMMs for Web Testing

Since UMMs are enhanced FSMs, FSM construction described in Section 4 can be reused as the first step to construct UMMs. The additional step involves assigning transition probabilities. Similar to the construction for Musa OPs described above, transition probabilities could be obtained by several methods, including: subjective evaluation based on *expert opinions*, *survey* of target customers, and *measurement* of actual usage. UMMs can be constructed based on existing access logs, using a combination of existing tools and internally implemented utility programs. However, as with most model construction activities, fully automated support is neither practical nor necessary. Human involvement is essential in making various modeling decisions, such as to extract UMM hierarchies and to group pages or links:

- Not every higher-level state needs to be expanded into lower-level models, because testing using lower-level model are to be performed selectively. Therefore, a threshold should be set up so that only the ones above it need to be expanded with their corresponding lower-level UMMs constructed.
- For traditional organizations, there is usually a natural hierarchy, such as university-school-department-individual, which is also reflected in their official web sites. There are generally closer interconnections as represented by more frequent referrals within a unit than across units. This natural hierarchy is used as the starting point for the hierarchies in these UMMs for web testing, which are later adjusted based on other referral frequencies.
- For links associated with very small link probability values, grouping them together to form a single link would significantly simplify the resulting model, and highlight the frequently used navigation patterns. A simple lower-level model for this group can be obtained by linking this single grouped node to all those it represents to form a one-level tree.
- Web pages related by contents or location in the overall site structure can also be grouped together to simplify UMMs.

Although any web page can be a potential entry point or initial state in an FSM or its corresponding Markov chain, one basic idea in statistical testing is to narrow this down to a few entry points based on their usage frequencies. The destinations of incoming links to a web site from external sources or start-ups are the entry points for UMMs. These links include URL accesses from dialog boxes, user bookmarks, search engine results, explicit links from external pages, or other external sources. All these accesses were recorded in the access log, and the analysis result for SMU/SEAS is summarized in the entry page report in Table III. For this web site, the root page “`/index.html`” outnumber other pages as the entry page by a large margin. In addition, these top entry pages are not tightly connected. These facts lead us to build a single set of UMMs [20] with this root page as the main entry node to the top-level Markov chain.

The issue with exit points is more complicated. Potentially any page can be the exit point, if the user decides to end accessing the web site. That is probably why

TABLE III
TOP ENTRY PAGES TO SMU/SEAS

Entry page	Occurrences
/index.html	18,646
/ce/index.html	2778
/co/cams/index.html	2568
/ce/smu/index.html	2327
/netech/index.html	2139
/disted/phd/index.html	1036
/co/cams/clemscam.html	963
/disted/index.html	878
/cse/index.html	813

no such exit page report is produced by any existing analyzer. This problem can be handled implicitly by specific usage sequences associated with specific test cases: The end of a usage sequence is the exit point from the UMM. This decision implies that frequently visited pages are also more likely to be the exit node than infrequently visited pages, which makes logical sense.

Figure 8 shows the top-level Markov chain of the UMM for the SMU/SEAS web site. The following information is captured and presented:

- Each node is labeled by its associated web file or directory name, and the total outgoing direct hits calculated. For example, `out = 2099` for the root node "`/index.html`," indicates that the total direct hits from this node to its children pages is 2099.
- Each link is labeled with its direct hit count, instead of branching probability, to make it easier to add missing branching information should such information become available later. However, relative frequencies and conditional branching probabilities for links originating from a given node can be deduced easily. For example, the direct hit count from the page "`/index.html`" to the page "`gradinfo.html`" is 314; and the conditional branching probability for this link is $314/2099 = 20.5\%$.
- Infrequent direct hits to other pages are omitted from the model to simplify the model and highlight frequently followed sequences. However, the omitted direct hits can be calculated by subtracting out direct hits represented in the diagram from the total "`out`" hits of the originating node. For example, there are direct hits to nine other pages from the root page "`/index.html`." The combined direct hits are: $2099 - 431 - 314 - 240 - 221 = 893$.

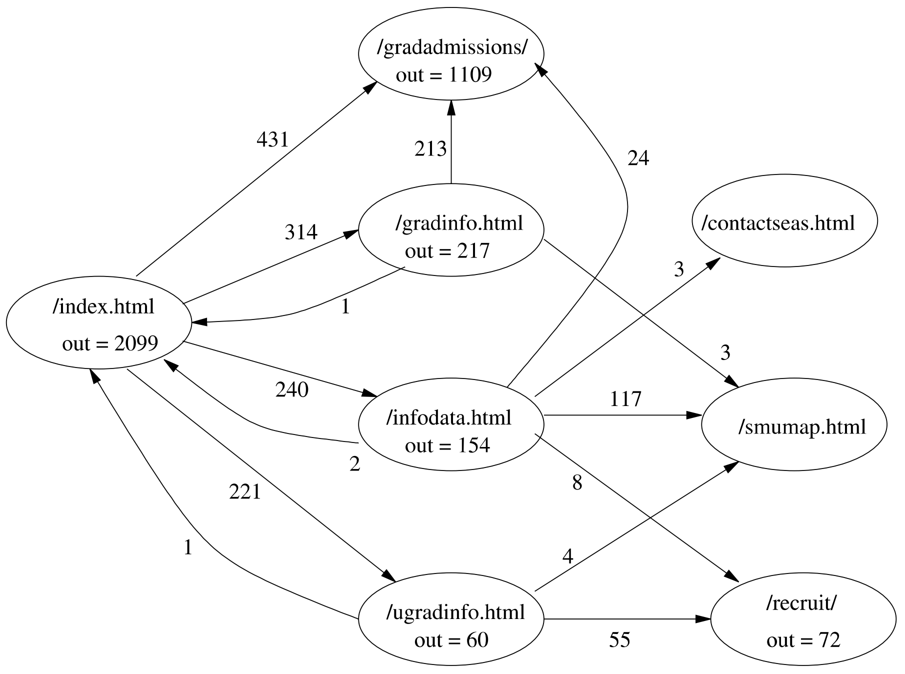

FIG. 8. Top-level UMM for SMU/SEAS.

Lower-level models are also produced for the nodes "`/gradadmission/`" and "`/recruit/`" in the top-level model. These models can be used to support our hierarchical strategy for statistical usage-based testing.

The test sensitization and outcome prediction are relatively simple and straightforward, similar to that for testing based on FSMs in Section 4. We can prepare all the input and specify the anticipated output and transitions ahead of time for most test situations. However, sometimes it would be hard to anticipate the input and the next state. Under such situations, dynamic test cases may be generated in the following way: From a current state, the transition or branching probabilities can be used to dynamically select the next state to visit, and sensitize it on the spot. Such dynamic test cases also have their own drawbacks, primarily in the reduced system performance due to the overhead to dynamically prepare and sensitize them.

The usage-based testing based on UMMs also yield data that can be used directly to evaluated the reliability of the implemented system, to provide an objective assessment of product reliability based on observation of testing failures and other related information. Unique to the usage of UMMs is that the failures can be associated with specific states or transitions. We can use such information to evaluate individual state reliability as well as overall system reliability, to extrapolate system reliability to different environments, and to identify high risk (low reliability) areas for focused reliability improvement.

5.5 Internal Validation via Cross-Reference Characterization

To validate the soundness of our integrated testing strategy above and its various decisions, we examined the cross-references recorded in the access log for the official pages of SMU/SEAS. The results are plotted in Fig. 9, which can be considered as a specific CPR introduced earlier in this section. As discussed in Section 4, embedded links in official pages are the primary responsibilities of the web site hosts, and consequently the focus of our study and the object of our analysis here. The sorted names of individual official pages are used as indexes in Fig. 9. Each point represents a cross-reference from a specific page indexed by its x-axis value to another

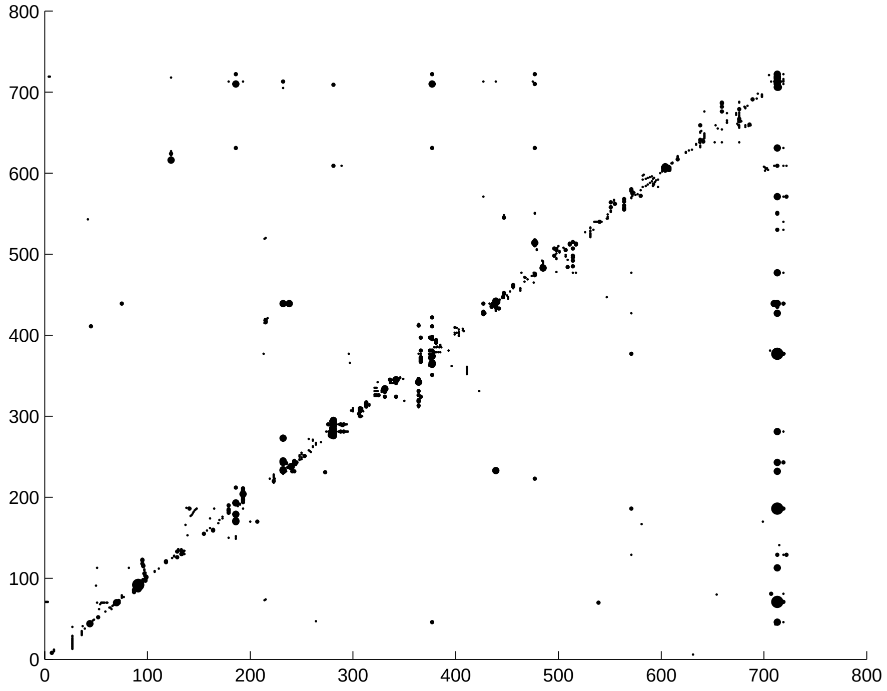

FIG. 9. Cross-references for sorted individual web pages.

specific page indexed by its y-axis value. A propositionally larger dot represents the number of duplicate cross-references. The references within a unit is then typified by the short distance between their indexes due to the same leading string in their names. The associated cross references would be represented by points close to the diagonal.

We examined the characteristics of high-level operations, and found that many items are loosely connected. This can be seen by the sparse points plotted in Fig. 9, particularly after we group some of the related pages together. Therefore, independent and individual probabilities used in Musa OPs are appropriate for high-level statistical web testing. For example, most of the pages with the highest access frequencies are index pages for individual academic units within SMU/SEAS, not tightly linked with each other and with little cross-references. Further analysis of expanded list of frequently visited pages follow similar patterns. In addition, the usage frequencies as well as the cross-reference frequencies are very unevenly distributed, as shown by the uneven distribution of points and masses in Fig. 9, thus justifying our use of statistical testing to focus on high-risk/high-leverage areas.

The exception to the loose connection above among the top access index pages is the hierarchical structure reflected, where there are numerous “up” and “down” references between the index page for SMU/SEAS and those for its individual departments and other academic units. In fact, there is a cluster of cross-references representing the common links followed from the site index page and other related pages to individual sub-sites or units, as represented by the cluster of points that form a vertical band to the right of Fig. 9. This can be tested using a high-level UMM to test the interaction between SMU/SEAS and its academic units.

While investigating the transition from top-level Musa OPs to middle-level UMMs, we noticed some natural clusters, which would be handled by individual UMMs. There is a close link and high cross-reference frequencies within a cluster but low visit frequencies across pages from different clusters. For example, within each academic department or unit, there are numerous cross-references but few across boundaries, as illustrated by the dominance of diagonal clusters in Fig. 9. Consequently, a UMM can be associated with each academic unit in statistical testing of this web site.

On the other hand, completely (or nearly completely) isolated operations can be tested by the top-level Musa OP alone, or if necessary, can bypass the middle level and go directly to the bottom-level model for further testing. This latter finding would require minor adjustments to our initially proposed approach and make our three-tiered testing strategy more flexible. In effect, we have modified our model to provide a bypass from top-level Musa OP to bottom-level structural testing.

6. Web Workload Characterization and Reliability Analyses

For software systems under normal operation or in testing, the execution results can be analyzed to assess test effectiveness and product reliability. In general, the failure information alone is not adequate to characterize and measure the reliability of a software system, unless there is a constant workload [27,34]. Due to the vastly uneven web traffic observed in previous studies [1,39], we need to measure both the web failures and related workload for reliability analyses. We next adapt existing techniques that characterize workload for traditional software systems and analyze their reliability to the web environment [52]. These measurement and analysis results also provide external validation or effectiveness assessment for our integrated testing strategy described in the previous section.

6.1 Defining Workload Measures for Reliability Assessment

The characteristics of the web environment discussed in Section 3 require us to measure actual web workload to ensure its satisfactory reliability instead of indiscriminately using generic measures suitable for traditional computation-intensive workload. The user focus and substantial amount of idle time during browsing sessions make any variation of execution time [34] unsuitable for web workload measurement. Similarly, the dominance of non-computational tasks also makes computational task oriented transactions [51] unsuitable for web workload measurement. Instead, other workload measures may be more suitable for characterizing workload at web sites.

From the perspective of web service providers, the usage time for web applications is the actual time spent by every user at the local web site. However, the exact time is difficult to obtain and may involve prohibitive cost or overhead associated with monitoring and recording dynamic behavior by individual web users [39]. One additional complication is the situation where a user opens a web page and continues with other tasks unrelated to the page just accessed. In this situation, the large gap between successive hits is not a reflection of the actual web usage time by this user. To approximate the usage time, we can consider the following workload measures [52]:

- *Number of hits, or hit count.* The most obvious workload measure is to count the number of hits, because (1) each hit represents a specific activity associated with web usage, and (2) each entry in an access log corresponds to a single hit, thus it can be extracted easily.

- *Number of bytes transferred, or byte count.* Overall hit count defined above can be misleading if the workload represented by individual hits shows high variability. Consequently, we can choose the number of bytes transferred, or byte count, as the workload measure of finer granularity, which can be easily obtained by counting the number of bytes transferred for each hit recorded in access logs.
- *Number of users, or user count.* User count is another alternative workload measure meaningful to the organizations that maintain the web sites and support various services at the user level. When calculating the number of users for each day, we treat each unique IP address as one user. So, no matter how many hits were made from the same computer, they are considered to be made by the same user. This measure gives us a rough picture of the overall workload handled by the web site.
- *Number of user sessions, or session count.* One of the drawbacks of user count is its coarse granularity, which can be refined by counting the number of user sessions. In this case, along with the IP address, access time can be used to calculate user sessions: If there is a significant gap between successive hits from the same IP address, we count the later one as a new session. In practice, the gap size can be adjusted to better reflect appropriate session identification for the specific types of web applications.

The number of user sessions per day may be a better measure of overall workload than the number of users, because big access gaps are typically associated with changes of users or non-web related activities by the same user. Each user who accesses the same web site from the same computer over successive intervals will be counted by user sessions, as long as such a gap exists in between. Even for a single user, a significant access gap is more likely to be associated with different usage patterns than within a single time burst. Therefore, by using user sessions, we can count the users' active contribution to the overall web site workload more accurately.

To summarize, the above measures give us workload characterization at different levels of granularity and from different perspectives. Hit count is more meaningful to web users as they see individual pages; byte count measures web traffic better; while number of users or sessions provide high-level workload information to web site hosts and content providers.

6.2 Measurement Results and Workload Characterization

On the average, each day for the SMU/SEAS web site is associated with 301.6 Mbytes, 29,345 hits, 2338 sessions, and 2120 users; each user is associated with 13.8 hits; each user session is associated with 11.6 hits; and each hit is associated with 10,279 bytes. We used the standard two-hour gap [32] to identify user sessions here.

The overall traffic at the KDE web site is significantly heavier than that for SMU/SEAS, with a daily average of 3563 Mbytes, 455,005 hits, 24,656 users, 29,029 s1 sessions, and 104,490 s2 sessions. Two different variations of session count were used in dealing with the KDE data: the same two-hour gap cut-off we used for the SMU/SEAS web site (labeled s1), and the 15 minutes cut-off more appropriate for dynamic pages (labeled s2) [39].

No matter which workload measure is used, the daily workload shows several apparent characteristics for both web sites, as follows:

- *Uneven distribution and variability*: The distribution is highly uneven and varies from day-to-day, as represented by the peaks and valleys in workload plots, which conforms to previously observed traffic patterns [1,21,39]. Among the four workload measures, daily bytes and daily hits show larger variability in relative magnitudes than daily users or daily sessions. This result indicates that although the number of users or user sessions may remain relatively stable, some users may use the web much more intensively than others, resulting in larger variations in detailed web site workload measurements over time. The relative differences for KDE tends to be smaller than that for SMU/SEAS, likely due to the heavier traffic by a larger and more diverse user population world-wide.
- *A periodic pattern that synchronize with error profile*, which is characterized by weekdays normally associated with heavier workload than during the weekends. This pattern seems to conform to the self-similar web traffic [12]. In addition, this periodic pattern are correlated or synchronized with daily error profile. This fact indicates that these workload measures are viable alternatives for web software reliability evaluation, because of the direct relation between usage and possible failures for the web site's source contents demonstrated in such synchronized patterns.
- *A long term stability for the overall trend*, which can be cross-validated by examining the trend over a longer period. All the workload measures traced over a year for SMU/SEAS showed long-term stability. This is probably due to the stable enrollment for SMU/SEAS and web site stability where no major changes were made to our web-based services over the observation period.

Of the four workload measures, hits, users, and sessions can be extracted from access logs easily and consistently. However, byte counting was somewhat problematic, because "byte transferred" field was missing not only for the error entries but also for many other entries. Further investigation revealed that most of these missing entries were associated with files or graphics already in the users cache ("file not modified," therefore no need to resend or reload). Since the total number of entries with missing bytes information represented about 15% of the total number of entries (hits), we simply used the rest to calculate the number of bytes transferred.

Workload measurements associated with periods shorter than calendar days, such as by the hour or by some short bursts, can also be used in reliability analysis. With time-stamped individual hits and related information available in the access log, such measurements can be easily obtained. As for reliability analysis, data with less variability, usually through data grouping or clustering, are generally preferable, because they typically produce more stable models that fit the observations better and provide better reliability assessments and predictions [41,47]. In our case, daily data have much less variability than hourly ones, yet give us enough data points for statistical analyses and model fitting. Consequently, we only use daily data in subsequent reliability analyses.

6.3 Analyzing Operational Reliability

As mentioned in Section 2, a workload profile with considerable variability, such as the web traffic for our two web sites studied here, is a clear indication that measuring failures alone over calendar time is not suitable for reliability analysis. The number of problems encountered per day is expected to be closely related to the usage intensity. When we combine the measurement results for the web failures, in this case type E errors extracted from the error log, and workload measured by the number of users, sessions, hits, and bytes transferred, we can perform analyses to evaluate web software reliability.

As observed earlier, the peaks and valleys in errors generally coincide with the peaks and valleys in workload. This close relationship between usage time and failure count can be graphically examined as in Fig. 10, plotting cumulative errors vs. cumulative bytes transferred over the observation period. An essentially linear relation can be detected between the two. Similar observations can be obtained if we plot cumulative errors vs. cumulative hits, users, or sessions.

This relationship can also be characterized by the daily failure rate, as defined by the number of errors divided by the workload measured by bytes transferred, hits, users, or sessions for each day. These daily failure rates also characterize web software reliability, and can be interpreted as applying the Nelson model [36] mentioned in Section 2 to daily snapshots. Table IV gives the range (min to max), the mean, and the standard deviation (std.dev.), for each daily error rates defined above. Because these rates are defined for different workload measurement units and have different magnitude, we used the relative standard error, or *rse*, defined as: $rse = std.dev./mean$, to compare their relative spread in Table IV. We also included the daily error count for comparison. All these daily error rates fall into tighter spread

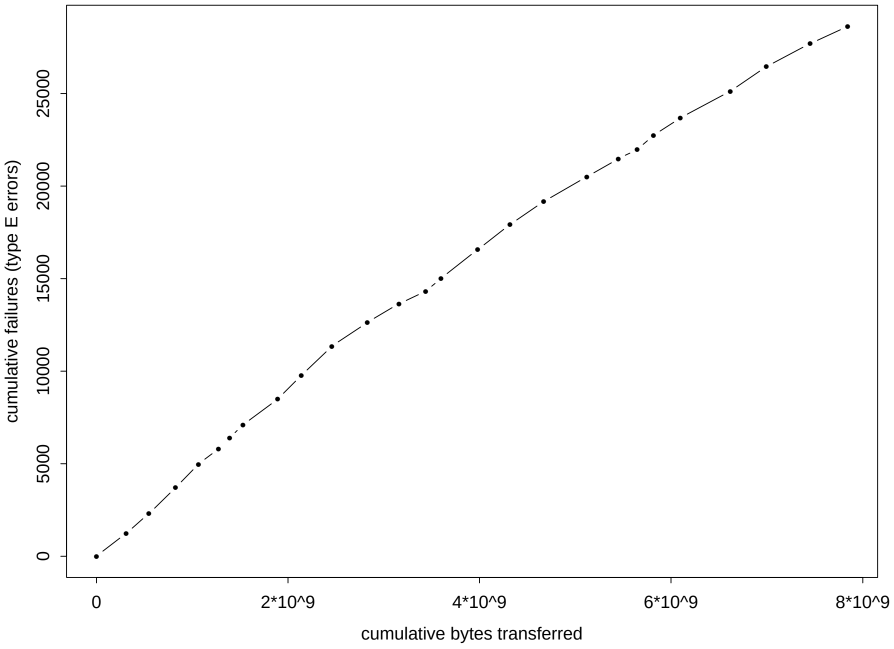

FIG. 10. Cumulative errors vs. cum. bytes transferred for SMU/SEAS.

TABLE IV
DAILY ERROR RATE (OR FAILURE RATE) FOR SMU/SEAS

Error rate	min	max	mean	std.dev.	rse
errors/bytes	2.35×10^{-6}	5.30×10^{-6}	3.83×10^{-6}	9.33×10^{-7}	0.244
errors/hits	0.0287	0.0466	0.0379	0.00480	0.126
errors/sessions	0.269	0.595	0.463	0.0834	0.180
errors/users	0.304	0.656	0.5103	0.0859	0.168
errors/day	501	1582	1101	312	0.283

than daily error count, which indicates that they provide more consistent and stable reliability estimates than daily error count alone.

Since individual web failures are directly associated with individual hits, we can use the Nelson model [36] to evaluate the overall web software reliability using failures and hits for the complete 26 days. This gives us the site software reliability of $R = 0.962$, or 96.2% of the individual web accesses will be successful. This model also give us an MTBF $=$ 26.6 hits, or averaging one error for every 26.6 hits.

TABLE V
DAILY ERROR RATE (OR FAILURE RATE) FOR KDE

Error rate	min	max	mean	std.dev.	rse
errors/bytes	3.608×10^{-6}	1.246×10^{-5}	7.210×10^{-6}	1.81×10^{-6}	0.251
errors/hits	0.04178	0.09091	0.05519	0.0117	0.211
errors/s1s	0.6335	1.4450	0.8648	0.189	0.219
errors/s2s	0.1665	0.4041	0.2403	0.0554	0.231
errors/users	0.7428	1.7060	1.0180	0.228	0.223
errors/day	15,510	44,160	25,330	6833	0.270

Modeling with other workload measures is also possible. For example, the above MTBF can be re-calculated for other workload units, giving us an MTBF = 273,927 bytes, an MTBF = 1.92 users, or an MTBF = 2.12 sessions. That is, this site can expect, on the average, to have a problem for every 273,927 bytes transferred, for every 1.92 users, or for 2.12 sessions. The web software reliability R in terms of these workload measures can also be calculated by the Nelson model. However, result interpretation can be problematic, because web failures may only be roughly associated with these workload measures. For example, because of the missing byte transferred information in the access logs for failed requests, the failures can only be roughly placed in the sequence of bytes transferred, resulting in imprecise reliability assessments and predictions. On the other hand, individual web failures may be roughly associated with certain users or user sessions through the particular hits by the users or within the sessions. In this case, each user or session may be associated with multiple failures, and appropriate adjustments to modeling results might be called for. For example, it might be more appropriate to separate failure-free sessions from sessions with failures, instead of comparing the number of failures in a session.

Table V gives the evaluation results of operational reliability for the KDE web site. Expectedly, the same patterns hold, i.e., all the daily failure rates fall into tighter bands than that for the daily errors, to give consistent and stable assessments of the operational reliability of this web site's contents. The overall reliability values are also roughly the same as that for SMU/SEAS. For example, on average, 5.76% of the hits would result in 404 errors, or the web site was 94.2% reliable as compared to 96.2% for SMU/SEAS.

6.4 Evaluating Potential Reliability Improvement

Under the idealized environment, the fault(s) that caused each observed failure can be immediately identified and removed, resulting in no duplicate observations of identical failures. This scenario represents the upper limit for the potential reliability improvement if we attempt to fix operational problems on-line or if we attempt to test the system and fix problems under simulated customer operational environment using our integrated testing strategy described in Section 5. This upper bound on reliability growth may not be attainable under many circumstances because of the large number of transient faults that usually take place whose origins are usually very difficult to be identified and removed because of their dependency on the context. Nevertheless, this upper limit gives us an idea about the potential reliability growth. Should quantitative information become available about the faults that are hard to fix, it can be used to fine tune the above limit to provide more accurate estimate of reliability growth potential.

This limit on potential reliability improvement can be measured by the reliability change (or *growth*) through the operational duration or testing process where such defect fixing could take place. Under the web application environment, each observed failure corresponds to a recorded type E error in the error log, and the idealized defect fixing would imply no more observation of any duplicate type E errors. In other words, failure arrivals under this hypothetical environment would resemble the sequence of unique type E errors extracted from the error log, which can be calculated by counting each type E error only once at its first appearance but not subsequently.

In general, reliability growth can be visualized by the gradual leveling-off of the cumulative failure arrival curve, because the flatter the part of the curve, the more time it takes to observe the next failure. To visualize this, we plotted in Fig. 11 cumulative unique failures versus different workload measurements we calculated above. Relative scale is used to better compare the overall reliability growth trends. The individual data points in the middle depict the failure arrivals indexed by cumulative hits. The (top) solid line depicts failure arrivals indexed by the cumulative bytes transferred. The (bottom) dashed line depicts failure arrivals indexed by the cumulative number of users. The user session measurement resulted in almost identical curve shape as that for the number of users, thus was omitted to keep the graph clean. As we can see from Fig. 11, there is an observable effect of reliability growth for this data, with the tail-end flatter than the beginning for all three curves.

Quantitative evaluation of reliability growth can be provided by software reliability growth models (SRGMs) we described in Section 2, which commonly assume instantaneous defect fixing [27,34]. In this chapter, we use a single measure, the purification level ρ [45] to capture this reliability change:

$$\rho = \frac{\lambda_0 - \lambda_T}{\lambda_0} = 1 - \frac{\lambda_T}{\lambda_0}$$

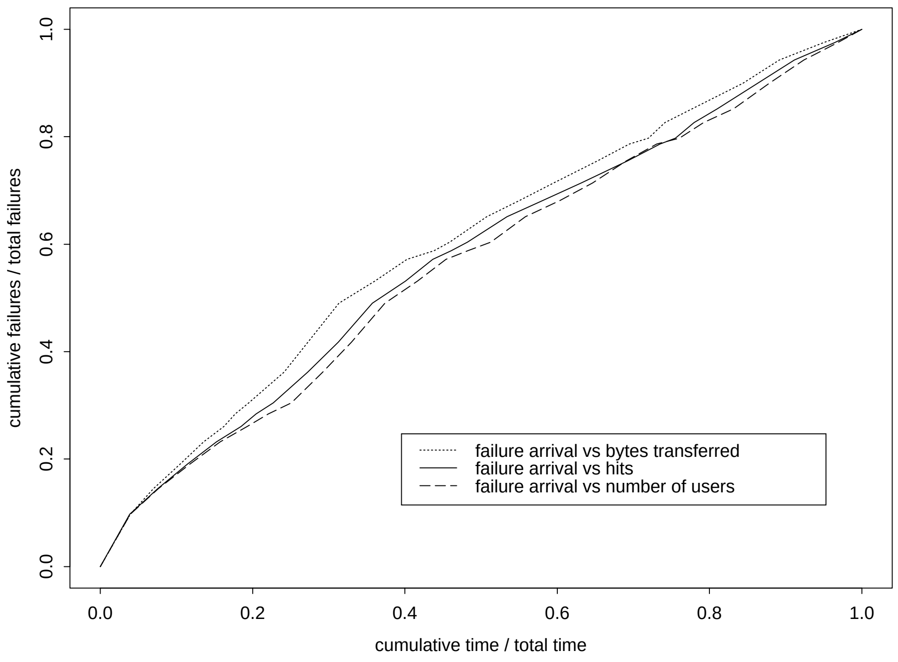

FIG. 11. Reliability growth comparison for different workload measures for SMU/SEAS.

where λ_0 and λ_T are the initial and final failure rates, respectively, estimated by a fitted SRGM. Complete elimination of all potential defects would result in $\rho = 1$, and no defect fixing would result in $\rho = 0$. Normal reliability growth is associated with ρ values ranging between these two extremes, with larger ρ values associated with more reliability growth. $(1 - \rho)$ gives us the ratio between λ_T and λ_0, or the final failure rate as a percentage of the initial failure rate.

We fitted the widely used Goel–Okumoto (GO) model [17] introduced in Section 2 to relate cumulative unique failures ($m(t)$) to cumulative workload measurements (t) in the formula:

$$m(t) = N\left(1 - \mathrm{e}^{-bt}\right).$$

Table VI summarizes these modeling results, giving estimated model parameters N and b, λ_0, λ_T, and ρ. The ρ values based on models using different workload measurements indicate that potential reliability improvement ranges from 57.9 to 74.8% in purification levels. In other words, effective web testing and defect fixing equiv-

TABLE VI
RELIABILITY MODELING RESULTS FOR SMU/SEAS

Time/workload measurement	Model parameters & estimates				Reliability growth ρ
	N	b	λ_0	λ_T	
bytes	3674	1.76×10^{-10}	6.45×10^{-7}	1.63×10^{-7}	0.748
hits	4213	1.38×10^{-6}	0.00583	0.00203	0.632
sessions	4750	1.42×10^{-5}	0.0675	0.0284	0.579
users	4691	1.60×10^{-5}	0.0752	0.0311	0.587

alent to 26 days of operation could have reduced the failure rate to between slightly less than half ($1 - 57.9\%$) and about one quarter ($1 - 74.8\%$) of the initial failure rate. Other SRGMs we tried also yield similar results: A significant reliability improvement potential exists if we can capture the workload and usage patterns in log files and use them to guide software testing and defect fixing.

We also repeated the assessment of reliability growth potential for the KDE web site. However, when we extracted the unique failures (unique 404 errors), we noticed an anomaly at the 24th day, which was associated with more than 10 times the maximal daily unique errors for all the previous days. Further investigation revealed that this is related to a planned beta release of the KDE product, when the web contents were drastically changed and many new faults were injected. Since our reliability growth evaluation is for stable situations where few or none new faults are injected, as is the assumption for all the software reliability growth models [27,34], we restricted our data to the first 22 days in this analysis.

Figure 12 plots the reliability growth evaluation we carried out for the KDE data. Among the five workload measures we used, bytes, hits, users, s1 and s2 sessions, all produced almost identical results in the reliability growth visualization, when we plotted relative cumulative unique errors against relative cumulative workload, similar to what we did in Fig. 11. The comparative visualization is omitted here because all the relative reliability growth curves would closely resemble the actual curve represented by the actual data points in Fig. 12. A visual inspection of Fig. 12 also revealed more degrees of reliability growth, or more bending of the data trend and fitted curve, than that in Fig. 11. Reliability growth potential as captured by ρ for the KDE web site ranged from 86.7 to 88.9% (with the model in Fig. 12 gave us $\rho = 87.1\%$). In other words, effective web testing and defect fixing equivalent to 22 days of operation could have reduced the failure rate to about 11 to 13% (calculated by $1 - \rho$) of the initial failure rate; or, equivalently, almost all the original problems could have been fixed.

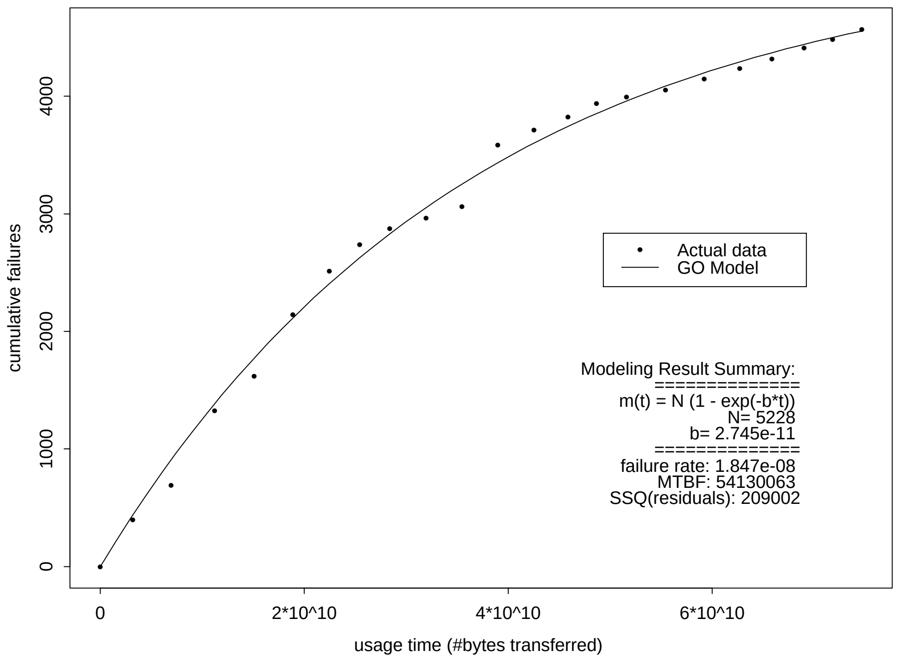

FIG. 12. A sample SRGM fitted to KDE data.

6.5 General Observations about Web Reliability and Test Effectiveness

As demonstrated in this section through case studies, web software reliability and test effectiveness can be evaluated based on information extracted from existing web server logs. Our key findings and related observations are summarized below:

- *Measure derivation and data extraction*: Four workload measures, bytes transferred, hit count, number of users, and number of sessions, were derived in this section for web workload characterization and reliability evaluation. Hit count, byte count, and user count can be easily extracted from access logs, due to their direct correspondence to access log entries and the data fields "bytes transferred" and "IP address." Session count computation may involve history information for individual users or unique IP addresses, but properly identified user sessions with appropriate time-out values can reflect web usage better than simply counting the users. Detailed failure data can be extracted from error logs. When such logs are not available, rough failure data can be extracted from access logs.

- *Assessing the operational reliability for web software*: When used with failure data to estimate failure rate or reliability, all four workload measures used in this section produced more consistent and stable reliability estimates than using daily errors alone. They offer reliability assessments from different perspectives, and each may be suitable for specific situations. For example, byte count might be more suitable for traffic measurement and related reliability interpretations; hit count might be more meaningful to web users as they browse individual pages; while number of users or sessions might be more suitable for high-level web site reliability characterization.
- *Assessing the potential for reliability improvement under effective testing*: Also demonstrated in both case studies is the significant potential for reliability improvement if defect can be fixed using our testing strategy described in Section 5. For www.seas.smu.edu, the failure rate could be reduced to 42.1–25.2% of the initial failure rate through such testing equivalent to 26 days of operation. Similarly for www.kde.org, the failure rate could be reduced to about 11–13% of the initial failure rate through testing equivalent to 22 days of operation; or, equivalently, almost all the original problems could have been fixed. These results provide external validation for our integrated testing strategy described in Section 5.
- *Some generalization beyond our two case studies*: Many of the results we obtained and patterns we observed concerning workload measurements for the SMU/SEA and the KDE web sites are remarkably similar to that for other Internet traffic [1,12,21,39], which indicates that web traffic characteristics have remained fairly stable for almost a decade. Although re-confirming these existing results and patterns is not our intention or our focus, this confirmation lends further validity to our primary purpose of using these measurements as part of the data input to evaluate web software reliability.

7. Conclusions and Perspectives

To summarize, a collection of appropriate testing techniques can be selected, adapted, and integrated to help us perform effective web testing and to ensure web reliability. As we demonstrated in this chapter through our case studies using the web sites for the School of Engineering and Applied Science, Southern Methodist University (SMU/SEAS) and for the large-scale open source KDE project, hierarchical testing of web-based applications is both viable and effective:

- The user focus of web-based applications is supported in this integrated strategy by testing functions, usage scenarios, and navigation patterns frequently used by web users under our top-tier usage models based on the list-like Musa operational profiles [33] as well as our middle-tier usage models based on Unified Markov Models (UMMs) [20,49].
- Individual web functions and internal components can be thoroughly exercised by using our bottom-tier models based on traditional coverage-based testing [6, 35], under the guidance of our upper-level usage models.
- Appropriate workload measures [52] can be extracted from access logs to capture the overall workload at different levels of granularity and from different perspectives. When used in conjunction with failure measurements, they can provide an objective and stable evaluation of web site operational reliability. A potentially significant reliability gain could be achieved under effective usage-based testing, leading to reliability growth ranging from 58 to 89% in purification levels [45] within a month of such improvement actions.
- The techniques and tools used by us for data collection, model construction and application, and result analysis and presentation can provide automated support that is essential for practical implementation and deployment of our strategy in industry.

There are some limitations to our approach, primarily in data availability and granularity, assumption of web stability in reliability evaluations, and lack of analyses based on detailed defect information. Related issues we plan to address in future studies include:

- *Overcoming data availability limitations*: Some information useful to support our integrated strategy is missing from existing web logs, such as the bytes transferred for failed or cached accesses, web navigation based on cached web contents at the users' side, ambiguity with "empty" referral information, other types of problems not isolated to the web server alone, etc. A logical next step is to search for alternative information sources to provide such information for more effective web testing and reliability improvement. The availability of such additional data will better support our integrated strategy.
- *Overcoming limitations of static web contents*: Our analyses are based on the default access logs of web servers, where dynamic errors concerning execution of dynamic web logic are not recorded. With more and more web sites providing dynamic information, dynamic logs will become widely available. We plan to expand and validate our approach on diverse web sites and make use of dynamic web logs in our future studies.

- *The impact of web site changes and related fault injections*: Our testing and reliability analyses in this chapter assumed the stability of the web sites under study, and our evaluation of reliability growth potential additionally assumed that none or few new faults were injected. Therefore, a direct generalization of this study is to study the impact of web changes and injection of new faults on test effectiveness and web reliability.
- *Detailed defect analysis and risk identification for reliability improvement*: Web error distribution is highly uneven, as shown in this chapter and in related studies [25,28,52]. To analyze them further, we plan to adapt detailed defect classification and analysis schemes such as ODC (orthogonal defect classification) [10] for traditional systems to the web environment. We also plan to apply appropriate risk identification techniques [46] and related models [45,53] to identify high-defect or low-reliability areas for focused web software reliability improvement.

In addition, we also plan to identify better existing tools, develop new tools and utility programs, and integrate them to provide better implementation support for our strategy. All these efforts should lead us to a more practical and effective approach to achieve and maintain high reliability for web applications.

Acknowledgements

The work reported in this chapter was supported in part by NSF grant 0204345 and THECB/ATP grant 003613-0030-2001. We thank our system administrator, Merlin Wilkerson, for his help in gathering and interpreting the web logs for the SMU/SEAS web site, and Dirk Mueller from the KDE project who helped us obtain and interpret the access logs for the KDE web site and deal with various data problems. We also thank Chaitanya Kallepalli and Omar Shahdad for their previous work in developing many of the Perl programs used in this study, and A. Güneş Koru and Zhao Li for their help and suggestions.

References

[1] Arlitt M.F., Williamson C.L., “Internet web servers: Workload characterization and performance implications”, *IEEE/ACM Trans. Networking* **5** (5) (Oct. 1997) 631–645.

[2] Avritzer A., Weyuker E.J., “The automatic generation of load test suites and the assessment of the resulting software”, *IEEE Trans. Software Engrg.* **21** (9) (Sept. 1995) 705–716.

[3] Bachiochi D.J., Berstene M.C., Chouinard E.F., Conlan N.M., Danchak M.M., Furey T., Neligon C.A., Way D., "Usability studies and designing navigational aids for the World Wide Web", *Computer Networks and ISDN Systems* **29** (8–13) (Sept. 1997) 1489–1496.
[4] Behlandorf B., *Running a Perfect Web Site with Apache*, second ed., MacMillan Computer Publishing, New York, 1996.
[5] Beizer B., *Software Testing Techniques*, second ed., International Thomson Computer Press, Boston, MA, 1990.
[6] Beizer B., *Black-Box Testing: Techniques for Functional Testing of Software and Systems*, John Wiley & Sons, Inc., New York, 1995.
[7] Black R., *Critical Testing Processes*, Addison–Wesley, Reading, MA, 2004.
[8] Boehm B., Basili V.R., "Software defect reduction top 10 list", *IEEE Computer* **34** (1) (Jan. 2001) 135–137.
[9] Bowers N., "Weblint: Quality assurance for the World-Wide Web", *Computer Networks and ISDN Systems* **28** (7–11) (May 1996) 1283–1290.
[10] Chillarege R., Bhandari I., Chaar J., Halliday M., Moebus D., Ray B., Wong M.-Y., "Orthogonal defect classification—a concept for in-process measurements", *IEEE Trans. Software Engrg.* **18** (11) (Nov. 1992) 943–956.
[11] Constantine L.L., Lockwood L.A.D., "Usage-centered engineering for web applications", *IEEE Software* **19** (2) (Mar. 2002) 42–50.
[12] Crovella M.E., Bestavros A., "Self-similarity in world wide web traffic: Evidence and possible causes", *IEEE/ACM Trans. Networking* **5** (6) (Dec. 1997) 835–846.
[13] Deo N., *Graph Theory with Applications to Engineering and Computer Science*, Prentice Hall, Englewood Cliffs, NJ, 1974.
[14] Frankl P.G., Weyuker E.J., "Testing software to detect and reduce risk", *J. Systems Software* **53** (3) (Sept. 2000) 275–286.
[15] Fromme B., "Web software testing: Challenges and solutions", in: *InterWorks'98*, Apr. 1998.
[16] Ghezzi C., Jazayeri M., Mandrioli D., *Fundamentals of Software Engineering*, second ed., Prentice Hall, Englewood Cliffs, NJ, 2003.
[17] Goel A.L., Okumoto K., "A time dependent error detection rate model for software reliability and other performance measures", *IEEE Trans. Reliability* **28** (3) (1979) 206–211.
[18] Howden W.E., "Functional testing", *IEEE Trans. Software Engrg.* **SE-6** (2) (1980) 162–169.
[19] IEEE, *IEEE Standard Glossary of Software Engineering Terminology*, Number STD 610.12-1990, IEEE, 1990.
[20] Kallepalli C., Tian J., "Measuring and modeling usage and reliability for statistical web testing", *IEEE Trans. Software Engrg.* **27** (11) (Nov. 2001) 1023–1036.
[21] Karagiannis T., Molle M., Faloutsos M., "Long-range dependence: Ten years of Internet traffic modeling", *IEEE Internet Computing* **8** (5) (Sept. 2004) 57–64.
[22] Karlin S., Taylor H.M., *A First Course in Stochastic Processes*, second ed., Academic Press, New York, 1975.
[23] Kitchenham B., Pfleeger S.L., "Software quality: The elusive target", *IEEE Software* **13** (1) (Jan. 1996) 12–21.
[24] Knuth D.E., *The Art of Computer Programming*, Addison–Wesley, Reading, MA, 1973.

[25] Li Z., Tian J., "Analyzing web logs to identify common errors and improve web reliability", in: *Proc. IADIS International Conference on E-Society, Lisbon, Portugal*, June 2003, pp. 235–242.

[26] Li Z., Tian J., "Testing the suitability of Markov chains as web usage models", in: *Proc. 27th Int. Computer Software and Applications Conf., Dallas, TX*, Nov. 2003, pp. 356–361.

[27] Lyu M.R., *Handbook of Software Reliability Engineering*, McGraw-Hill, New York, 1995.

[28] Ma L., Tian J., "Analyzing errors and referral pairs to characterize common problems and improve web reliability", in: *Proc. 3rd International Conference on Web Engineering, Oviedo, Spain*, July 2003, pp. 314–323.

[29] Miller E., *The Website Quality Challenge*, Software Research, Inc., San Francisco, CA, 2000.

[30] Mills H.D., "On the statistical validation of computer programs", Technical Report 72-6015, IBM Federal Syst. Div., 1972.

[31] Mills H.D., Basili V.R., Gannon J.D., Hamlet R.G., *Principles of Computer Programming: A Mathematical Approach*, Alan and Bacon, Inc., Boston, MA, 1987.

[32] Montgomery A.L., Faloutsos C., "Identifying web browsing trends and patterns", *IEEE Computer* **34** (7) (July 2001) 94–95.

[33] Musa J.D., *Software Reliability Engineering*, McGraw-Hill, New York, 1998.

[34] Musa J.D., Iannino A., Okumoto K., *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.

[35] Myers G.J., *The Art of Software Testing*, John Wiley & Sons, Inc., New York, 1979.

[36] Nelson E., "Estimating software reliability from test data", *Microelectronics and Reliability* **17** (1) (1978) 67–73.

[37] Peper M., Hermsdorf D., "BSCW for disabled teleworkers: Usability evaluation and interface adaptation of an Internet-based cooperation environment", *Computer Networks and ISDN Systems* **29** (8–13) (Sept. 1997) 1479–1487.

[38] Pfleeger S.L., Hatton L., Howell C.C., *Solid Software*, Prentice Hall, Upper Saddle River, NJ, 2002.

[39] Pitkow J.E., "Summary of WWW characterizations", *World Wide Web* **2** (1–2) (1999) 3–13.

[40] Porter A.A., Siy H., Votta L.G., "A review of software inspections", in: Zelkowitz M.V. (Ed.), *Advances in Computers*, vol. 42, Academic Press, San Diego, CA, 1996, pp. 39–76.

[41] Schneidewind N.F., "Software reliability model with optimal selection of failure data", *IEEE Trans. Software Engrg.* **19** (11) (Nov. 1993) 1095–1104.

[42] Shneiderman B., *Designing the User Interface: Strategies for Effective Human–Computer Interaction*, Addison–Wesley, Reading, MA, 1987.

[43] Stallings W., *High Speed Networks and Internets: Performance and Quality of Service*, second ed., Prentice Hall, Upper Saddle River, NJ, 2001.

[44] Thayer R., Lipow M., Nelson E., *Software Reliability*, North-Holland, New York, 1978.

[45] Tian J., "Integrating time domain and input domain analyses of software reliability using tree-based models", *IEEE Trans. Software Engrg.* **21** (12) (Dec. 1995) 945–958.

[46] Tian J., "Risk identification techniques for defect reduction and quality improvement", *Software Quality Professional* **2** (2) (Mar. 2000) 32–41.

[47] Tian J., "Better reliability assessment and prediction through data clustering", *IEEE Trans. Software Engrg.* **28** (10) (Oct. 2002) 997–1007.

[48] Tian J., *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*, John Wiley & Sons, Inc. and IEEE CS Press, Hoboken, NJ, 2005.

[49] Tian J., Lin E., "Unified Markov models for software testing, performance evaluation, and reliability analysis", in: *4th ISSAT International Conference on Reliability and Quality in Design, Seattle, WA*, Aug. 1998.

[50] Tian J., Ma L., Li Z., Koru A.G., "A hierarchical strategy for testing web-based applications and ensuring their reliability", in: *Proc. 27th Int. Computer Software and Applications Conf. (1st IEEE Workshop on Web-based Systems and Applications), Dallas, TX*, Nov. 2003, pp. 702–707.

[51] Tian J., Palma J., "Test workload measurement and reliability analysis for large commercial software systems", *Ann. Software Engrg.* **4** (Aug. 1997) 201–222.

[52] Tian J., Rudraraju S., Li Z., "Evaluating web software reliability based on workload and failure data extracted from server logs", *IEEE Trans. Software Engrg.* **30** (11) (Nov. 2004) 754–769.

[53] Tian J., Troster J., "A comparison of measurement and defect characteristics of new and legacy software systems", *J. Systems Software* **44** (2) (Dec. 1998) 135–146.

[54] Trivedi K.S., *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, second ed., John Wiley & Sons, Inc., New York, 2001.

[55] Vatanasombut B., Stylianou A.C., Igbaria M., "How to retain online customers", *Commun. ACM* **47** (6) (June 2004) 65–69.

[56] Whittaker J.A., Thomason M.G., "A Markov chain model for statistical software testing", *IEEE Trans. Software Engrg.* **20** (10) (Oct. 1994) 812–824.

[57] Zelkowitz M.V., "Role of verification in the software specification process", in: Yovits M.C. (Ed.), *Advances in Computers*, vol. 36, Academic Press, San Diego, CA, 1993, pp. 43–109.