The Journal of Systems and Software xxx (2009) xxx–xxx

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Multi-faceted quality and defect measurement for web software and source contents ☆

Zhao Li, Nasser Alaeddine, Jeff Tian *

Southern Methodist University, Computer Science and Engineering Dept., Dallas, Texas 75275, USA

ARTICLE INFO

Article history:
Received 31 October 2008
Received in revised form 7 March 2009
Accepted 28 April 2009
Available online xxxx

Keywords:
Quality and reliability
Defect measurement and analysis
Web software and contents
Dynamic web applications
Web application development and maintenance

ABSTRACT

In this paper, we examine external failures and internal faults traceable to web software and source contents. We develop related defect and quality measurements based on different perspectives of customers, users, information or service hosts, maintainers, developers, integrators, and managers. These measurements can help web information and service providers with their quality assessment and improvement activities to meet the quality expectations of their customers and users. The different usages of our measurement framework by different stakeholders of web sites and web applications are also outlined and discussed. The data sources include existing web server logs and statistics reports, defect repositories from web application development and maintenance activities, and source files. We applied our approach to four diverse websites: one educational website, one open source software project website, one online catalog showroom for a small company, and one e-Commerce website for a large company. The results demonstrated the viability and effectiveness of our approach.

© 2009 Elsevier Inc. All rights reserved.

1. Introduction

The World Wide Web (WWW), or simply the web, is a way of accessing information and conducting personal and business transactions over the Internet. Initially intended as an information highway for publishing static hypertexts, the web is increasingly supporting dynamic applications, with non-trivial computations performed at run-time. The increased complexity also demands disciplined methodologies and measurements for web application development and maintenance (Fenton and Pfleeger, 1996; Pfleeger et al., 2002). With the prevalence of the web and people's reliance on it in today's society, ensuring its satisfactory reliability is also becoming increasingly important (Bélanger et al., 2006; Offutt, 2002).

Quality in software is generally characterized by the absence of observable problems and satisfaction of user expectations, which can also be related to some internal characteristics of the software product and its development process (Chrissis et al., 2006; Pfleeger et al., 2002; Tian, 2005). A quantitative measure of quality from a user's perspective is product *reliability*, or how likely a user can use the software without encountering a failure (Musa, 1998; Thayer et al., 1978). Alternatively, as the internal causes for observed failures, software faults can be measured and analyzed to provide an internal assessment of software quality and help drive quality assurance and improvement activities (Fenton and Pfleeger, 1996; Kan, 2002).

Building upon our previous work adapting traditional quality measurement and improvement techniques to the web domain (Alaeddine and Tian, 2007; Kallepalli and Tian, 2001; Li, 2008; Ma and Tian, 2007; Tian and Ma, 2006; Tian et al., 2004), we construct a multi-faceted framework for measuring the software and web contents quality and defects. After a brief review of basic concepts in Section 2 and an examination of web challenges in Section 3, we introduce our measurement framework from several different perspectives in subsequent sections:

- In Section 4, defect and usage data from in-field web applications captured in web server logs are used to evaluate website operational reliability in providing the requested contents or services and the potential for reliability growth under effective testing. This external quality measurement and related failure measurement are from a user's perspective.
- Internal defects are measured and characterized using some fault count and density metrics adapted to fit the web environment in Section 5, which are directly related to problem fixing effort therefore meaningful to website maintainers and support personnel.
- In Section 6, long-term trend in web application quality meaningful to website owners, managers, business clients, and long-term customers are captured in coarse-grain defect metrics

☆ This work is supported in part by NSF Grants CCR-0204345 and IIP-0733937.
* Corresponding author. Tel.: +1 214 768 2861; fax: +1 214 768 3085
E-mail address: tian@engr.smu.edu (J. Tian).
URL: http://www.lyle.smu.edu/~tian (J. Tian).

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.04.055

based on data extracted from web statistics reports produced by existing tools.

- For computation-rich dynamic Internet applications such as e-Commerce, we provide measurement of underlying software faults in the "deep" source view (He et al., 2007) of defect based on data from web development and maintenance activities captured in defect repositories. Quality and defect measurement from this perspective is particularly meaningful to web application developers, integrators, and project managers. We also present a procedure to convert the collective defect data from both defect repositories and web server logs between the failure view and fault view, effectively linking this internal view of web quality to quality perception of users of such dynamic web applications. These topics are covered in Section 7.

Case studies applying this approach to four diverse websites are included throughout the paper to demonstrate its viability and effectiveness, followed by an overall summary of our measurement framework and its usage by different stakeholders of web sites and web applications in Section 8. Conclusions and perspectives are presented in Section 9.

2. Basic concepts, definitions, and measurements

Various terms related to problems or defects are commonly used in discussing software quality and reliability. Several standard definitions (IEEE, 1990) 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.

Failures and faults are collectively referred to as *defects*, which are tracked during software development and maintenance activities with the help of various tools. Software quality is typically measured by how long the software can run before encountering a failure from an external user's perspective or by the number of faults in the software that need to be fixed from an internal perspective. The former is captured by software *reliability*, formally defined as the probability of failure-free operations for a software system for a given period of time or a given input set under a specific environment (Musa, 1998; Thayer et al., 1978); while the later is captured by defect or fault count and density (Kan, 2002).

To help us evaluate the current reliability and reliability change over time, input domain reliability models (IDRMs) and time domain software reliability growth models (SRGMs) are commonly used (Musa, 1998; Thayer et al., 1978). IDRMs provide a snapshot of the current product reliability. For example, if f failures are observed for n execution instances, the estimated reliability R according to the Nelson IDRM (Nelson, 1978) is:

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

where r is the failure rate, which is also often used to characterize reliability.

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. For example, in the widely used Goel–Okumoto SRGM (Goel and Okumoto, 1979), the expected cumulative failures, $m(t)$, over time t is given by the formula:

$$m(t) = N(1 - e^{-bt})$$

where the model constants N and b can be estimated from the observation data.

Both defect counts and defect density have been used as an internal measure of software quality (Kan, 2002). Defect density is typically defined as:

$$\text{Defect density} = \frac{\text{total number of defects}}{\text{size of the software}} \times 100\%$$

where the number of defects is the total defects identified against a particular software entity during a particular time period. The size of the software is a normalizer that allows comparisons between different software entities. For traditional software, size is typically counted either in Lines of Code (LOC) or Function Points (FP) (Fenton and Pfleeger, 1996; Kan, 2002). Defect density can also be used to assess the effectiveness of development and quality improvement activities and, more importantly, to help identify defect prone components for focused quality improvement.

3. Characterizing web challenges and opportunities

As a new type of software application, the web presents many challenges. We next characterize web quality problems and examine opportunities to address these problems.

3.1. Web characteristics and problem definition

Compared to traditional software systems, web applications have several unique characteristics:

- *Size, population, and diversity*: the sheer size of the web, the diverse hardware/software/network configurations and support facilities, the massive user population, and the varied usage patterns need to be considered in measuring and ensuring web quality.
- *Evolving nature*: taking advantage of the web infrastructure that makes it easy for developers and maintainers to make additions, updates, or changes, the web is continually evolving, with ever-changing contents, functions, and services.

All parts of this complex and ever-changing system need to function well to satisfy quality and other expectations of its massive and diverse user population. Based on these unique characteristics, we define *web failures* as the inability to correctly obtain or deliver information, such as documents or computational results, requested by web users. The *reliability* for web applications can be defined as the probability of failure-free web operation completions, which are typically among the top concerns for web users (Bélanger et al., 2006; Offutt, 2002). Web software or source content failures are related to the *acquisition* of the requested information by web users because of problems such as missing or inaccessible files, trouble with starting JavaScript, etc. These failures and related internal faults that cause these failures are closely identified with the site-specific web-based functions and services, and will be the focus of our study. We exclude host and network failures that prevent the *delivery* of requested information to web users over the Internet and user and browser problems at the client's end, because they are beyond the control of web service or content providers.

The above problems are only made worse with the introduction and wide-spread usage of dynamic web applications, which have become increasingly integrated into business strategies for companies. While the content of static web pages is fixed, the content of dynamic pages is computed at run-time by the server according to user input as well as the state on the server, such as the date, time, user, location, or session information. Dynamic sites are highly intertwined with the environment (browsers, operating systems,

UNCORRECTED PROOF

database engines, web servers, and interfaces to onsite or offsite applications). In addition to HTML documents and other static components, dynamic web applications integrate a wide range of technologies, including various language modules, data manipulation languages, and databases. Dynamic web applications can be broken down into components associated with four layers: presentation, business logic, backend connectors/interfaces/proxies, and data layers. In addition, some important components also involve all layers, including state, cache, and environment/configuration/deployment management. An important category of problems for dynamic web applications is the missing or incorrect functional behaviors and corresponding problem sources, which are also the focus of our study.

3.2. Information sources for problem analysis and quality measurement

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*. In fact, monitoring web usage and keeping various logs are necessary to keep a website operational. Therefore, we would only incur minimal additional cost to use these logs for web quality and defect measurement.

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. Specific information in access logs typically includes: the identity of the machine making the request, the user id used in authentication, the user identity, a time-stamp, the complete first line of the HTTP request in quotes, the HTTP response code, total number of bytes transferred, the referrer, and the agent. Error logs typically include details about the problems or web failures encountered preceded by an observation time-stamp.

For dynamic web applications used in e-Commerce and other business applications, more disciplined development methodologies are adopted to coordinate development effort that spans beyond a few individuals over a short period of time. Defects are formally logged, tracked, and resolved during the development and maintenance activities, which provide additional valuable data. We will refer to them as *functional defects* or incorrect or missing implementations of requirements, which may only be detected in development or system maintenance cycle but not through analyzing web server logs. The recorded defect attributes usually include: project name, defect summary, detail description, date detected, assigned to, expected date of closure, detected by, severity, defect ID, software build version, and any supplemental notes. The data are stored in centralized repositories where each defect record corresponds to a unique software fault.

In addition to the web server log data and defect repositories above, some coarse-grain data and source files can also be used to measure web quality and defects. More and more websites have adopted web log analysis software into their web administration tool collection to generate web statistics reports. Web server source files are at least partially accessible to the public due to the blending of source files with navigation facilities in HTML and other documents. Therefore, we will also use such statistics reports and source files for our quality and defect measurement.

3.3. Websites for our case studies

As a direct extension of our previous work adapting traditional quality measurement and improvement techniques to the web domain (Alaeddine and Tian, 2007; Kallepalli and Tian, 2001; Li, 2008; Ma and Tian, 2007; Tian and Ma, 2006; Tian et al., 2004), we started with the website for the Engineering School at Southern Methodist University (SMU/SEAS, www.seas.smu.edu) that utilizes the popular Apache Web Server and shares many common characteristics of websites for educational institutions. Then, we gradually expanded our case studies to include three diverse websites, including: the open source KDE project website (www.kde.org), a online catalog showroom for a small company (hereafter labeled SC), and an e-Commerce website for a large company in the telecommunications industry (hereafter labeled LTC).

All these four websites use access logs to keep record of daily activities. We extracted failure information in the form of entries with specific response code for our defect and reliability analysis. In addition, functional defects from development and maintenance activities are available for LTC, which are "deep"-analyzed to examine the collective defects from both these sources in a systematic way. Because of the data sensitivity issue, the only error logs, source files, and long-term data available to us are for SMU/SEAS, restricting our detailed defect analysis, defect density analysis, and long-term coarse-grain reliability analysis to this website only.

SMU/SEAS server log data used in our detailed defect and reliability analysis cover 26 consecutive days in 1999. In addition, we also performed defect density analysis based on a snapshot of source files from 2006, and coarse-grain long-term reliability analysis based on web statistics reports from 2004 to 2006.

Through work with open source projects (Koru and Tian, 2004), we obtained web logs from the KDE project. KDE is a network transparent contemporary desktop environment for UNIX workstations developed and maintained by a large open group consisting of hundreds of software engineers from all over the world. KDE website provides project information, supports download of released documentation and software, and provides online development facilities. Changes are continuously committed to the website in order to provide the developers and users with the most up-to-date information. We selected data over 31 days in 2003 for our subsequent analyses.

SC website is a commercial website using *HP Proliant* as the web server with *Redhat ES*. It uses a combination of scripts and static pages, with the contents of most requested pages dynamically generated by the scripted language on-the-fly. PHP, Javascript and Perl are used extensively to generate dynamic pages. Those scripts are based on the open source software package "Gallery", with modifications made for SC's business purposes. The access log data used in this study cover 31 days in 2006.

LTC website is a deployed online ordering application for a large telecommunication company that processes a couple of million requests a day. It provides a wide range of services, including: browse available telecom services, view accounts information, submit inquiries, order new services, change existing services, view order status, and request repair. It consists of hundreds of thousands of lines of code and utilizes IIS 6.0 (Microsoft Internet Information Server). It was developed using Microsoft technologies such as ASP, VB scripts, and C++. Both access logs and defect data repository for LTC from 2007 are used in this study.

4. Web failures and reliability: a user's perspective

Data about application failures attributed to software and source contents can be extracted from existing server logs. When placed into the context of the corresponding durations or usage instances, these failures and related reliability give an overall picture of the quality of the website or web application from a user's perspective.

4.1. Preliminary analysis of common failures

Common web problems or failure types are listed in Table 1, together with the actual observations for SMU/SEAS (Kallepalli and

UNCORRECTED PROOF

Table 1
Generic failure types and the recorded failures by type for SMU/SEAS.

Type	Description	#failures
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

Tian, 2001). For SMU/SEAS, type E failures ("file does not exist") are the most common type of problems, accounting for 93.1% of the total recorded failures. They usually represent bad links, which can be further analyzed to assess web content reliability. The majority of these bad links are from internal links, including mostly URLs embedded in some web pages and sometimes from pages used as start-ups at the same website (Tian and Ma, 2006). Only a small percentage of these failures are from other websites (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 content reliability based on local actions.

From the HTTP response code in error logs, we can extract the general failure information for other websites. For example, type E failures in error logs are equivalent to access log entries with a response code 404. The access logs for the KDE website for the 31 days recorded more than 14 million hits, of which 793,665 resulted in failures. 785,211 hits resulted in response code 404, which accounted for 98.9% of all the failures. The next most reported failure type was of response code 408, or "request timed out", which accounted for 6225 or 0.78% of all the failures.

We also observed similar failure distributions for SC and LTC. For example, the access logs for SC recorded more than 402,939 hits, of which 102,654 resulted in failures. Close to 100% of them, or 102,647, are those with response code 404. For the LTC website, 7.20% of the recorded accesses resulted in failures, with 404 failures making up 98.1% of them, followed by 500 (server internal error) at 0.97%, All these results justified our focus on 404 failures.

4.2. Measuring workload and assessing operational 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 (Musa, 1998). Due to the vastly uneven web traffic observed in our previous studies (Tian et al., 2004; Tian and Ma, 2006), we need to measure both the web failures and usage time for reliability analyses. From the perspective of web service providers, the usage time for web applications is the actual time spent by every user at the local website. However, the exact time is difficult to obtain and may involve prohibitive cost. To approximate the usage time, the following workload measures are used:

- *Number of hits.* This is the most obvious choice 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,* which can be easily obtained by tallying the number of bytes transferred for each hit recorded in access logs. If the workload represented by individual hits shows high variability, it would measure workload more accurately than hit count.
- *Number of users,* which gives us a rough picture of the overall workload meaningful to the organizations that support various services at the user level.
- *Number of user sessions.* If there is a significant gap between successive hits from an IP address, we count the later one as a new session, with the gap size adjusted to better reflect appropriate session identification for the specific web applications.

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 website hosts and content providers. No matter which workload measure is used, the daily workload distribution varies greatly for the four websites we studied, with the workload measurements synchronized with the observed failures.

This synchronized relationship between workload and failures can be characterized by the daily failure rate, as defined by the number of failures 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 (Nelson, 1978) to daily snapshots. Table 2 gives the range, the mean, and the standard deviation (std.dev), for each daily failure rate defined above. Reliability R can be calculated from Table 2 as $R = 1 - r$. For example, the average web content reliability is $R = 0.9621$, or 96.2% success rate for individual web accesses, or averaging one failure for every 26.6 hits ($1/R$). Similar reliability measures with other measurement units and for the other websites we studied were also calculated to give their respective users an overall picture of the expected reliability.

Because these failure rates are defined for different measurement units and have different magnitude, we used the relative standard error, or *rse*, defined as: $rse = \frac{std.dev}{mean}$, to compare their relative spread in Table 2. All these daily failure rates fall into tighter spread than daily failure count, which indicates that they provide more consistent and stable reliability estimates than daily failure count alone. Expectedly, the same patterns hold for the other websites we studied.

4.3. Evaluating potential reliability improvement

Under the idealized environment, the fault that caused each observed failure can be immediately identified and removed, result-

Table 2
Daily failure rate r for SMU/SEAS.

Failure rate	Min	Max	Mean	Std.dev	rse
Failures/bytes	2.35×10^{-6}	5.30×10^{-6}	3.83×10^{-6}	9.33×10^{-7}	0.244
Failures/hits	0.0287	0.0466	0.0379	0.00480	0.126
Failures/sessions	0.269	0.595	0.463	0.0834	0.180
Failures/users	0.304	0.656	0.5103	0.0859	0.168
Failures/day	501	1582	1101	312	0.283

UNCORRECTED PROOF

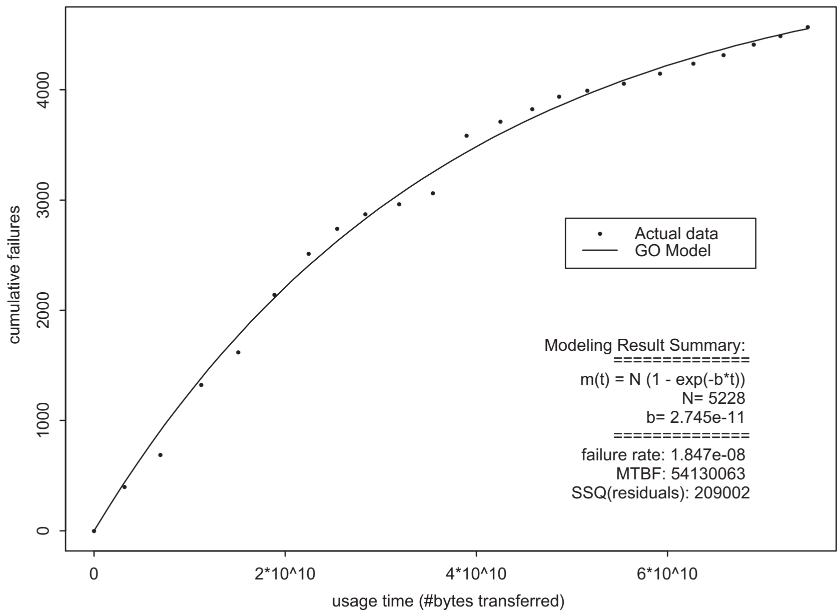

Fig. 1. A sample SRGM fitted to KDE data.

ing in no duplicate observations of identical failures. This upper limit on potential reliability improvement can be measured by the reliability change (or *growth*) through the duration when such defect fixing could take place. For this analysis, each observed failure corresponds to a recorded 404 (or type E) failure, and the idealized defect fixing would imply no more observation of any duplicate failures. In other words, failure arrivals under this hypothetical environment would resemble the sequence of unique failures extracted from the web server logs, which can be calculated by counting each failure only once at its first appearance but not subsequently. Quantitative evaluation of the reliability growth potential can be captured by the purification level ρ (Tian, 2005) defined as:

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

where λ_0 and λ_T are the initial and final failure rates, respectively, estimated by a fitted SRGM. A larger ρ value is associated with more reliability growth, with $\rho = 1$ associated with complete elimination of all potential defects, and $\rho = 0$ associated with no defect fixing at all. $(1 - \rho)$ gives us the ratio between λ_T and λ_0, or the final failure rate as a percentage of the initial failure rate.

Fig. 1 plots the reliability growth evaluation using Goel–Okumoto (GO) model (Goel and Okumoto, 1979) for the KDE data to relate cumulative unique failures to cumulative workload measured by the bytes transferred over 22 days. It gave us a reliability growth potential of $\rho = 87.1\%$. When we used other usage time measurements, including hits, users, sessions, ρ values for KDE fall into a tight range between 86.7% and 88.9%. 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% of the initial failure rate; or, equivalently, almost all the original problems could have been fixed. Similar results were also obtained for the other websites we studied.

5. Web faults and defect density: a maintainer's perspective

Although there is a general causal relation between faults and failures, it is not a one-to-one relation. Consequently, there is a practical reason for us to analyze faults and failures separately due to the different results expected, in addition to their different purposes and perspectives.

5.1. Fault measurement and analysis

Each identified software or content fault, in this case, the dominant individual missing files for our four websites, needs to be fixed by either providing the missing file or fixing the broken link. To website maintainers, identifying, measuring and analyzing these faults would be directly meaningful. Unlike failure observations that vary with usage instances, the underlying faults should remain the same unless drastic development or maintenance effort is applied. Therefore, for fault measurement, an overall count, distribution, and related analysis would be used instead of the time dependent failure and reliability analyses we performed in the previous section. The missing files can be simply identified as the unique failures in web server logs, counting each one only for its first appearance while ignoring all the duplicate entries pointing to the same missing file.

Table 3 gives the top impact faults for LTC, where the top five missing file faults contribute to 91.39% of the total 404 failures. For SMU/SEAS, 2913 missing files caused 28655 404 failures. The average number of requests per missing file ranged from 9.84 for SMU/SEAS to 209 for SC. We can also examine the distribution of different types of faults (different missing file types in this case) and the failures they cause, such as in Table 4 for SMU/SEAS. We can see that the missing file fault distribution and corresponding failure distribution are quite different, although a general positive correlation exists between the two. To compare faults across different websites or across different subsites, clusters, or subsets, we need to normalize them by their size, to produced a normalized measure similar to defect density used in traditional software systems.

5.2. Web defect density measurement

Because the web is hyperlink-driven, with its complexity closely identified with inter-connectivity between files, we define our defect density for the web environment as:

Table 3
Top 404 failure producing faults for LTC.

HTTP fault	Failures	% 404 Failures
/images/dottedsep.gif	5805	32.46
/images/gnav_redbar_s_r.gif	3687	20.62
/images/gnav_redbar_s_l.gif	3537	19.78
/includes/css/images/background.gif	2593	14.50
/includes/css/nc2004style.css	721	4.03

Table 4
Faults and failures by file type for SMU/SEAS.

File type	Faults	% of total	Failures	% of total
Directory	1135	38.96	4425	15.44
.html	943	32.37	3656	12.76
.gif	287	9.85	12,489	43.58
.ico	125	4.29	849	2.96
.jpg	106	3.64	1323	4.62
.ps	55	1.89	209	0.73
.pdf	44	1.51	237	0.83
.doc	32	1.10	78	0.27
.txt	26	0.89	32	0.11
.class	25	0.86	4913	17.15
Cumulative	2778	95.37	27,491	98.45
Total	2913	100	28,655	100

$$\text{Defect density} = \frac{\text{number of broken hyperlinks}}{\text{total number of hyperlinks}} \times 100\%$$

To obtain the total number of hyperlinks and identify the broken ones, all embedded hyperlinks need to be extracted and validated. We separated the hyperlinks into two categories: *in-site* hyperlinks that point to resources inside the same web server, and *out-site* hyperlinks that point to resources outside. The size of the web and the huge number of embedded hyperlinks demand automatic instead of manual hyperlink extraction and validation. We implemented an automated web defect density evaluation tool, optimized to minimize the network traffic by the following:

- Eliminate most of the need to send in-site hyperlink request for validation by using a hash table. For each hyperlink-able objects in the web server, a corresponding hyperlink is created and stored in a hash table. In-site hyperlink validation looks up the hash table first. It sends a HTTP request only if the hyperlink cannot be found in the hash table. This eliminates more than 95% of the internal HTTP requests.
- HEAD method is used when sending HTTP requests for out-site hyperlinks, because the returned "head" information contains enough information for hyperlink validation. In most circumstance, it reduces more than 99% of the web traffic by avoiding requests for the whole contents the hyperlinks point to.

The SMU/SEAS website, with 165,150 source files as of 2006, is used as the case study for web defect density measurement (Li, 2008). The results are given in Table 5, showing a apparent difference between the in-site and out-site web defect densities. Changes to resources that the out-site hyperlinks point to are made by different teams thus highly unpredictable to local website maintainers, resulting in more broken hyperlinks and significantly higher defect density for out-site hyperlinks than for the in-site category.

Table 5
Defect density measurement for SMU/SEAS.

	In-site	Out-site	Overall
# hyperlinks	1,113,620	41,112	1,154,732
# broken hyperlinks	23,686	6935	30,621
Defect density	2.1%	16.9%	2.7%

Table 6
Defect density measurement for SMU/SEAS subsites.

Defect density	In-site (%)	Out-site (%)	Overall (%)
JAVA 5	0.1	1.8	0.1
The rest	12.2	31.0	14.1

In this case study, we also identified half of the source files as from a self-contained subsite in SMU/SEAS. Further investigation reveals that the subsite is a copy of online Java 5 reference. Therefore, we separated SMU/SEAS into two subsites: Java 5 online reference subsite, with 82,807 files and 944,251 hyperlinks, and the rest, with 82,343 files and 210,481 hyperlinks. The results are given in Table 6, consistently showing the quality superiority of in-site over out-site hyperlinks. The results also indicate quality difference for contents developed by different groups: the subsite developed by SMU/SEAS staff members and individuals has much higher defect density than the Java 5 subsite developed by Sun Microsystems Inc. Our web defect density metric captures and quantifies such expected quality differences.

6. Long-term reliability: a business perspective

Both web defect density measure and web reliability measure using log files only give us a snapshot of internal quality or reliability over a relatively short period of time. Information sensitivity and size of source files and log files also make them difficult to obtain and use over extended periods of time. Therefore, we need to explore new ways to perform quantitative web long-term reliability analysis that can help web site owners and managers evaluate organization's capability to produce quality software and drive continuous improvement (Chrissis et al., 2006; Tian, 2005), which would also be of interests to their business clients and long-term customers.

Fortunately, more and more websites have adopted web log analysis software into their web administration tool collection, use it by default, and publish reports online. For instance, Analog, one of the most popular web administration tools, analyzes web server log files, extracting such items as client's IP addresses, URL paths, processing times, user agents, referrers, etc., and grouping them to produce various reports. Because these reports are in summary format without sensitive information, their accessibility is less problematic. The web statistics reports generally include the following information:

- *General Summary* contains some overall statistics about the log data being analyzed, including the number of requests, the number of requests for pages, the number of distinct hosts, and the amount of data transferred in bytes.
- Several reports about requested contents, including statistics about downloaded files (Request Report), directories for the downloaded files (Directory Report), file types (File Type Report), file sizes (File Size Report), HTTP status codes of all requests (Status Code Report).
- Several variations of client/user characterization, including statistics about organizations the client computers registered under (Organization Report), Internet domains for client computers (Domain Report), pages linked to local files (Referrer Report), servers those referrers were on (Referring Site Report), operating systems used by visitors (Operating System Report), and search terms people used to find the site (Search Word Report).

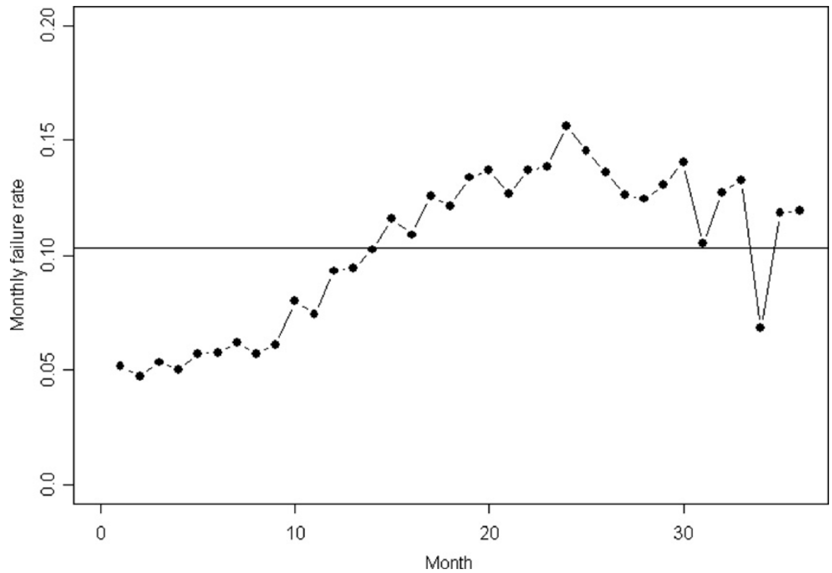

Fig. 2. Monthly failure rate for SMU/SEAS.

For long-term reliability analysis, web usage and failure information needs to be extracted from these statistics reports. We developed a web statistics report parser for this purpose (Li, 2008). Monthly reports from 2004 to 2006 produced by Analog for SMU/SEAS were processed to calculate monthly reliabilities in Fig. 2. Similar to in Section 4, only requests returning status code 404 are considered as failures, and the same operational reliability definition based on Nelson model (Nelson, 1978) is used to normalize monthly failures by corresponding hits. Fig. 2 indicates that the failure rate of website SMU/SEAS became worse over time from year 2004 to 2005, which was accompanied by the migration of the website to new design and functionalities.

7. Functional defects and their impact: development view

For dynamic web applications that have become increasingly integrated into business strategies for companies, an important category of problems is the functional defects that may only be detected in development or maintenance cycle but not through analyzing web server logs. These problems would require a "deeper" analysis (He et al., 2007) of problematic behavior and problem sources. The analysis results can be merged with those from analyzing web server logs to produce comprehensive and meaningful results for dynamic web applications.

7.1. Functional defects and collective fault view

Each recorded defect in defect repositories corresponds to a unique software fault in the dynamic web application. Table 7 summarizes the distribution of such functional faults for LTC, together with their corresponding HTTP categories. We found that only 10.63% of the recorded problems from the defect-tracking tool were also found in the server web logs as missing files. This insignificant overlap between data from defect repository and from web server logs indicates that we need both to produce comprehensive and meaningful results for such dynamic web applications.

To build the collective fault view, we need to merge functional defect data with those extracted from web server logs. As described in Section 5, failures in web server logs can be easily transformed into faults by identifying unique causes, or individual missing files for this case. Then, the missing file faults can be directly merged with the functional faults to build the collective fault view. We need to normalize the missing file and functional fault classes by the total number of defects after removing duplicate entries. Table 8 shows the collective fault view for LTC.

Table 7
Functional faults discovered for LTC.

Fault class	HTTP categories	% of total
Service/system interfaces	200/300	33.13
Graphical user interface	200/300	22.89
Code logic, compu. & algo.	200/300	20.48
Missing files	404	10.63
Missing verbiage	200/300	9.04
Missing links	200/300	2.63
Cache	200/300	1.20
Session/cookies	200/300	0.00
Concurrence/multi. users	200/300/400/500	0.00
Env./config./deployment	200/300/400/500	0.00
Operational behavior	200/300	0.00
HTTP failures except 404	400/500	0.00
Database	200/300	0.00
All	200/300/400/500	100.00

7.2. Functional defect impact analysis and collective failure view

If we evaluate the potential impact of functional faults based on defect severity and likely usage scenarios, we could obtain corresponding partial defect data in the failure view as well. When combined with defect data from web server logs, it give us a collective failure view, and provides data input for our focused testing and reliability improvement (Alaeddine and Tian, 2007).

Besides the raw defect data, the fault exposure list and the operational profile need to be constructed to help us assess fault impact. The functional defects are divided into categories based on

Table 8
Top classes of faults (collective fault view) for LTC.

Class of faults	% of total
Missing files - HTTP 404 failures	33.64
System/services interface	25.34
Graphical user interface	17.50
Logic, computation, and algorithm	15.67

Please cite this article in press as: Li, Z., et al. Multi-faceted quality and defect measurement for web software and source contents. J. Syst. Software (2009), doi:10.1016/j.jss.2009.04.055

Table 9
Defect exposure list.

Potential impact	Description	Weight
Showstopper	Prevents the completion of a central requirements	100%
High	Affects a central requirement and there is a workaround	70%
Medium	Affects non-central requirement and there is no workaround	50%
Low	Affects non-central requirement for which there is a workaround	20%
Exception	Affects non-conformity to a standard	5%

Table 10
Operational profile and number of transactions for LTC.

Operation	Probability	# of transactions
New order	0.10	588
Change order	0.35	2058
Move order	0.10	588
Order status	0.45	2646

the potential impact, and weight assigned to each of these categories by domain experts in Table 9. The operational profile (OP) is a quantitative characterization of how a software is or will be used in field by target customers and users (Musa, 1998), such as given by the first two columns of Table 10 based on available customer usage data. Each recorded functional defect is associated with a specific transaction or operation in the OP and a specific defect type. In practical applications, the combination of OP operation and defect type will uniquely identify recorded defects. The steps involved in this fault-failure mapping are:

1. Find the number of hits per server per day and calculate the total number of transactions based on the data from web access logs. For LTC, the number of hits was 235,142 per server per day with an estimated 40 hits per transaction on average. Therefore, the number of transactions per server per day is 235,142/40 = 5880.
2. Multiple the total number of transactions calculated above by the defined operational profile to obtain the number of transactions processed every day for each operation, which yields the results in Table 9 for LTC.
3. Multiply the number of transactions for each operation calculated above by the defined exposure list to calculate the failure frequency (impact) of the specific fault identified by the operation in OP and the defect type. Table 11 shows the failure view of the order status for LTC, and similar results can be obtained for other operations to build the complete failure view of the defects.

These steps map individual functional faults into potential failure instances, effectively providing an assessment of fault impact under this usage environment defined by the operational profile and product internal structure reflected in the fault exposure list.

We finally merge and sort information about missing file failures and functional failures to generate the fault impact ranking based on failure frequency. Table 12 shows the top individual faults for LTC ranked by their corresponding failure frequencies. We noticed that a large number of failures were caused by a small number of faults with high failure frequencies. By fixing these faults, total failures can be reduced dramatically. For example, by fixing the top 6.8% faults, the total failures were reduced by about 57%. The corresponding reliability improved from 0.9356 to 0.9723. Similarly, when we progressively focus on top impact faults, a overwhelming share of total failures can be eliminated (Alaeddine and Tian, 2007).

Table 11
Failure view of order status for LTC.

Operation	Potential impact	Weight (%)	# Transactions	Failure frequency
Order status	Showstopper	100	2646	2646
Order status	High	70	2646	1852
Order status	Medium	50	2646	1323
Order status	Low	20	2646	529
Order status	Exception	5	2646	132

8. Measurement framework and its usage

Our overall measurement framework and its four major components are summarized in Table 13. Within this general framework, our four measurement alternatives are derived from four different perspectives to offer their own unique benefits to different stakeholders:

- Quality from a user's view is measured by web failures and reliability. Overall failure data extracted from web server logs can be examined for trend and distribution. Operational reliability can be measured by the failure rates, or number of failures per operational instances measured by the number of requests, bytes transferred, users, or sessions. In addition, potential for reliability improvement can be measured by fitting unique failure data over operational instances to existing SRGMs. All these measurements are directly meaningful to web users and can help web contents and service providers in their maintenance activities.
- The internal quality from web maintainer's point of view is measured by web defect density based on source file analysis. It can help drive web quality improvements by focusing on areas of high defect density for defect fixing and other maintenance activities. Its use can help us remove defects before additional customers encounter related problems. Therefore, this measure is also indirectly meaningful to external customers and users.
- Quality from web host's and manager's long-term business view is measured by long-term web reliability based on analysis of available web statistics reports, which also reflects perceived website quality by business clients or long-term customers.
- Quality from developer's view for dynamic web applications is measured by functional faults extracted from defect repositories. These defects discovered during web application development and maintenance activities reflect problems with extensive backend facilities and dynamic contents in providing the web capabilities and services to target customers. The insignificant overlap we observed between data from defect repositories and from web server logs led us to use both in our collective fault and failure analysis to produce comprehensive and meaningful results for such applications. The procedure we developed to map functional faults to operational failures allows us to prioritize overall defect fixing effort in the most cost effective way in terms of delivered reliability improvement. This reliability improvement would directly benefit customers and users of such dynamic web applications (dwa.user in Table 13).

UNCORRECTED PROOF

Table 12
Individual fault impact ranking for LTC.

Rank	Response code	Fault	Failure frequency
1	404	/images/dottedsep.gif	5805
2	404	/images/gnav_redbar_s_r.gif	3687
3	404	/images/gnav_redbar_s_l.gif	3537
4	200/300	Order status – sys interface – showstopper	2646
5	404	/includes/css/images/background.gif	2593
6	200/300	Change order – logic – showstopper	2058
7	200/300	Order status – sys interface – high	1852
8	200/300	Order status – logic – high	1852
9	200/300	Change order – logic – high	1441
10	200/300	Change order – user interface – high	1441
11	200/300	Order status – sys interface – medium	1323
12	200/300	Order status – logic – medium	1323
13	200/300	Order status – user interface – medium	1323
14	200/300	Change order – sys interface – medium	1029
15	200/300	Change order – logic – medium	1029
16	200/300	Change order – user interface – medium	1029
17	404	/includes/css/nc2004style.css	721

Table 13
Overall summary of web quality and defect measurements.

Measurement	Perspective	Data sources	Case studies
Failures & Reliability	customer/user/info.seeker	Server logs	All four
Defect density	Maintainer	Source files	SMU/SEAS
Long-term reliability	host/manager/bus.client	statistics reports	SMU/SEAS
Functional faults/failures	Development/dwa.user	Defect repository & server logs	LTC

The evolving nature of the web requires us to focus on measuring maintenance. All four measurements in this paper capture web maintenance information, either as the coarse-grain summary information for long-term reliability measurement, source files and embedded hyperlinks as maintenance targets for defect density measurement, and in-field usage and problems in web server logs for the other two above. In addition, functional defects from development and maintenance activities for dynamic web applications are used for functional fault and failure measurement. All our measurements are defined for web maintenance at different levels of granularity, with functional defect measurement also applicable to web application development activities.

The data availability largely determines the applicability of our measurements to different websites and applications: sensitive user information is recorded in web server log files, making them available only to authorized analysts but not to the general public. Similar to defect data for traditional software systems, web functional faults are only accessible to web development and maintenance teams and parties they authorize. But unlike source code of traditional software systems, web server source files are at least partially accessible to the public due to the blending of source files with navigation facilities in HTML and other documents. Web statistics reports contain no sensitive data and are sometimes published online as part of the website.

Once the data are obtained, measurement and analysis activities performed in this paper, including data extraction, processing, modeling, and result analysis, are all supported by software tools and facilities we implemented, effectively removing one of the major obstacles to the implementation of our measurement strategy. The remaining implementation issues and related cost include: (1) the large storage space and processing required for reliability measurement, (2) some online hyperlink validation overhead for defect density measurement, and (3) additional input from users and experts in the form of operational profiles and fault exposure assessments used to convert functional faults to operational failures in functional fault measurement and related failure analysis. In practical applications, the data availability and implementation cost issues need to be resolved by balancing the expected benefit from such measurement and analysis activities against additional cost and effort needed to obtain the required data and to produce the desired analysis results.

To use our measurement framework, we recommend a customized approach based on stakeholder identification and related analysis of stakeholder concerns. For information seekers or casual users, the only meaningful defect-related measurement of web quality would be its overall reliability. They would only be concerned about how likely their requests will be handled without a problem, or that the information they seek is acquired and delivered. For business clients or long-term customers, long-term reliability and the overall reliability trend would be of interests too, in addition to the current reliability of the web site or the web application. In addition, they may be indirectly interested in knowing the defect density and number and/or density of functional defects because such internal defects have a direct impact on the observed failures and it takes time and resources to fix these problems. Therefore, a cost-benefit analysis might be needed to balance their desire for higher quality web sites and web applications against the additional cost it might incur.

For people on the producer side, the failure and reliability measurements should also be their concern because their direct linkage to the perceived web site or web application quality by customers, users, and business clients. The long-term viability of such web-based businesses or services depends on these measurements over time, as also reflected in the long-term reliability. However, more directly and for specific projects or for a short period of time, the internal defect related measures such as defect density, and additionally functional defects for dynamic web applications, are needed because their direct relationship to defect fixing cost and their important role in process control and project management. The defect impact analysis describe in this paper would bridge the gap between this internal and external perspectives of web site and web application quality.

Please cite this article in press as: Li, Z., et al. Multi-faceted quality and defect measurement for web software and source contents. J. Syst. Software (2009), doi:10.1016/j.jss.2009.04.055

Therefore, depending on different stakeholders different roles, responsibilities, and concerns, an appropriate subset of measures defined in this paper can be selected. Then data sources can be identified, measurement activities can be carried out, measurement results can be analyzed, including the possible "deep-analysis" for dynamic web applications, and feedback and recommendations can be provided. For example, information seekers or casual users would be satisfied to know that the overall reliability of target web sites are sufficiently high to allow them to continue using these sites in the future. Clients and long-term customers may be encouraged to work together with web site maintainers and managers to fix relevant and high-impact problems identified through defect density and long-term reliability measurement and analysis to ensure and improve web reliability and long-term survivability of web sites and web-based businesses. For dynamic web applications, the measurement and analysis of both functional defects related to underlying database and business logic and non-functional defects related to web layer will enable development teams to focus on high-impact faults to prioritize limited resources to achieve maximal reliability improvement. Such improved reliability would ensure customer satisfaction and long-term business viability.

9. Conclusions and perspectives

The general benefit of performing web quality measurement is to provide quantitative quality assessments, identify problematic areas, and drive continuous improvement. Towards this end, we defined appropriate measurements based on different stakeholders' perspectives: quality from a user's view is measured by web failures and reliability based on log file analysis, which are directly meaningful to web users and can help web contents and service providers in their maintenance activities. The internal quality from web maintainer's point of view is measured by web defect density based on source file analysis, which can be used to help drive web quality improvements by focusing on areas of high defect density for defect fixing and other maintenance activities. Quality from web host's and manager's long-term business view is measured by long-term web reliability based on analysis of available web statistics reports, which also reflects long-term customer's perception of website quality. Quality from developer's view for dynamic web applications is measured by functional faults extracted from defect repositories, and the procedure we developed to map them to operational failures allows us to prioritize overall defect fixing effort in the most cost effective way in terms of delivered reliability improvement that is meaningful to target customers and users.

With the measures defined above, we also identified the data sources and obtained necessary data for four diverse web sites: one academic website SMU/SEAS, one open source development website KDE, one online catalog showroom LC for a small company, and one commercial website LTC for a large telecommunications company with extensive dynamic contents for e-Commerce applications. The measurement results and the overall analysis results were provided as feedback to web site owners and other stakeholders. The impact of using these measurements and analysis results were also described, such as the accelerated defect discovery and reliability improvement in Section 4 and significant failure reduction by focusing on high-impact faults in Section 7. The positive results from these case studies demonstrated the applicability and effectiveness of our approach.

These measurements have the potential to form an synergistic measurement framework, with possible extensions to include other quality measurements and analysis techniques, for a wide variety of web contents and service providers to measure and improve web quality to better satisfy their customers. The initial success allows us to continue expanding this work to related activities to help our existing partners measure and improve the quality for their web sites and their dynamic web applications. It also gives us the opportunity to engage new partners in similar activities that expanded the application environments as well as important issues being addressed. We are also working to extend the software tools and facilities we implemented for measurement and analysis activities described in this paper to form a tool suite of enhanced capability and usability to help our current and future partners achieve their goals of web quality improvement and customer satisfaction.

Acknowledgement

The work reported in this paper was supported in part by NSF Grants CCR-0204345 and IIP-0733937. We thank Merlin Wilkerson and Dirk Mueller for their help in gathering and interpreting the web logs for the SMU/SEAS and the KDE websites respectively. We thank the anonymous reviewers for their constructive suggestions, particularly in summarizing our measurement framework and outlining its customized usage by different stakeholders.

References

Alaeddine, N., Tian, J., 2007. Analysis of anamalies and failures in dynamic web applications. In: Proc. 11th IASTED Int. Conf. on Software Engineering and Applications, pp. 385–290.
Bélanger, F., Fan, W., Schaupp, L.C., Krishen, A., Everhart, J., Poteet, D., Nakamoto, K., 2006. Web site success metrics: addressing the duality of goals. Communications of the ACM 49 (12), 114–116.
Chrissis, M.B., Konrad, M., Shrum, S., 2006. CMMI: Guidelines for Process Integration and Product Improvement, second ed. Addison-Wesley/Pearson Educationr, Boston, MA.
Fenton, N., Pfleeger, S.L., 1996. Software Metrics: A Rigorous and Practical Approach, second ed. PWS Publishing, Boston, MA.
Goel, A.L., Okumoto, K., 1979. A time dependent error detection rate model for software reliability and other performance measures. IEEE Transactions on Reliability 28 (3), 206–211.
He, B., Patel, M., Zhang, Z., Chang, K.C.-C., 2007. Accessing the deep web. Communications of the ACM 50 (5), 94–101.
IEEE, 1990. IEEE Standard Glossary of Software Engineering Terminology. No. STD 610.12-1990. IEEE.
Kallepalli, C., Tian, J., 2001. Measuring and modeling usage and reliability for statistical web testing. IEEE Transactions on Software Engineering 27 (11), 1023–1036.
Kan, S.H., 2002. Metrics and Models in Software Quality Engineering, 2/e. Addison-Wesley, Reading, MA.
Koru, A.G., Tian, J., 2004. Defect handling in medium and large open source software projects. IEEE Software 21 (4), 54–61.
Li, Z., 2008. Web reliability analysis and improvement. Ph.D. thesis, Southern Methodist University, Dallas, Texas, U.S.A.
Ma, L., Tian, J., 2007. Web error classification and analysis for reliability improvement. Journal of Systems and Software 80 (6), 795–804.
Musa, J.D., 1998. Software Reliability Engineering. McGraw-Hill, New York.
Nelson, E., 1978. Estimating software reliability from test data. Microelectronics and Reliability 17 (1), 67–73.
Offutt, J., 2002. Quality attributes of web applications. IEEE Software 19 (2), 25–32.
Pfleeger, S.L., Hatton, L., Howell, C.C., 2002. Solid Software. Prentice Hall, Upper Saddle River, New Jersey.
Thayer, R., Lipow, M., Nelson, E., 1978. Software Reliability. North-Holland, New York.
Tian, J., 2005. Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement. John Wiley & Sons Inc. and IEEE CS Press, Hoboken, New Jersey.
Tian, J., Ma, L., 2006. Web testing for reliability improvement. In: Zelkowitz, M.V. (Ed.), Advances in Computers, vol. 67. Academic Press, San Diego, CA, pp. 177–224.
Tian, J., Rudraraju, S., Li, Z., 2004. Evaluating web software reliability based on workload and failure data extracted from server logs. IEEE Transactions on Software Engineering 30 (11), 754–769.

Jeff (Jianhui) Tian received a B.S. degree in Electrical Engineering from Xi'an Jiaotong University in 1982, an M.S. degree in Engineering Science from Harvard University in 1986, and a Ph.D. degree in Computer Science from the University of

UNCORRECTED PROOF

Maryland in 1992. He worked for the IBM Software Solutions Toronto Laboratory between 1992 and 1995 as a software quality and process analyst. Since 1995, he has been with Southern Methodist University, Dallas, Texas, now as an Associate Professor of Computer Science and Engineering, with joint appointment at the Dept. of Engineering Management, Information and Systems. His current research interests include software testing, measurement, reliability, safety, complexity, and applications in commercial, web-based, telecommunication, and embedded software and systems. He is a member of IEEE and ACM. Q1

UNCORRECTED PROOF

Please cite this article in press as: Li, Z., et al. Multi-faceted quality and defect measurement for web software and source contents. J. Syst. Software (2009), doi:10.1016/j.jss.2009.04.055