Accelerated Defect Discovery and Reliability Improvement through Risk-Prioritized Testing for Web Applications

Jeff Tian*
Southern Methodist University, Dallas, Texas, USA.

* Corresponding author. Tel.: +1 214-768-2861; email: tian@smu.edu
Manuscript submitted September 21, 2022; accepted November 28, 2022.
doi: 10.17706/jsw.18.3.159-171

Abstract: In this paper, we present a risk-prioritized testing strategy aimed at accelerating defect discovery and reliability improvement for Web applications. This strategy is based on analyzing defect rate or density using Web logs and defect data from development activities to produce a prioritized list for testing. We empirically compare it against simulated coverage-based testing for five websites from diverse Web application domains, including academic, open source project, small business catalog showroom, large e-Commerce application, and social networking websites. The results from comparing their respective defect discovery profiles demonstrate the superiority of risk-prioritized testing in accelerating defect discovery and reliability improvement.

Keywords: Defect and risk profiles, quality and reliability, testing, Web applications and logs

1. Introduction

High *quality* software and Web applications are usually characterized by the absence of *failures*, or observable external problems, and satisfaction of user expectations [1]. Quality defined this way can be quantified by *reliability*, which is the probability of failure-free operations for a specific time period or input set under a specific environment [2, 3]. In addition to reliability, user expectations for high quality for Web applications may also include usability, security, performance, etc. [4].

Testing plays a critical role in assuring software reliability by detecting and removing defects through the process of executing software, observing its behavior and behavioral deviations (failures), and analyzing its artifacts to locate and fix underlying faults [5]. Most traditional testing techniques attempt to cover major external functions or product components, with the implicit assumption that higher coverage means higher quality [6, 7]. Alternately, product reliability goals can be used as an objective and quantitative criterion to stop testing and to evaluate testing techniques by their delivered reliability [3–8]. Usage-based statistical testing techniques under an environment that resembles actual usage by target customers are typically used for this latter purpose [3, 9, 10]. For Web applications, there are various existing data sources, which can be exploited to derive effective and efficient testing approaches to address quality needs of end users.

Under tight time and resource constraints for modern software systems characterized by their increased size and complexity, compounded by the diversity in their usage and associated rare conditions that trigger system failures, the above testing techniques may be pushed to the limit of their effectiveness and practicality [11–13]. In addition, most Web applications are multi-layered, with many diverse components

scattered over different locations and layers, and with the components themselves as well as the infrastructure evolving continually. Therefore, other novel approaches need to be explored, including test-case prioritization schemes based on product features and characteristics [14, 15] and risk-based or risk-prioritized testing technique in this paper. In our approach, we first identify high-risk areas – areas where faults are more likely to be exposed and/or area more likely to hide more defects. Armed with such knowledge, prioritized testing can then be performed, focusing on the high-risk areas for more effective fault removal and reliability improvement.

One primary goal for testing is to detect as many defects as possible, and as early as possible, to enable their removal in timely follow-up activities that can be initiated for defect fixing and reliability improvement [16, 17]. Therefore, testing effectiveness and efficiency can be evaluated by the extent that this goal can be achieved by examining the number of defects discovered against the amount of time or resources used to detect these defects. In this paper, we use the defect discovery profiles, or the number of defects discovered plotted against time spent or resource used, to compare the effectiveness and efficiency of risk-prioritized testing against other commonly used testing strategies.

2. Background

This work builds upon and extends our previous work on usage-based and risk-prioritized Web testing [1, 18–21], as well as automated Web defect information extraction and problem analysis for focused reliability and usability improvement [22, 23]. We started our initial research in this direction with an academic website, the Engineering School website at Southern Methodist University (SMU/SEAS, http://www.seas.smu.edu). An important reason for this is our ability to have full access to the Web server logs and statistics produced by existing tools, as well as access to most of the source contents. To validate and generalize our initial results, we gradually expanded our case studies to include four other websites, while striving to include diverse websites of different characteristics. Therefore, our risk-prioritized testing described in this paper has been applied in five websites from diverse Web application domains, including:

- the Engineering School website at Southern Methodist University mentioned above, hereafter labeled as SMU/SEAS,
- the open source project KDE website, hereafter labeled as KDE,
- an online catalog showroom website for a small company, hereafter labeled as SCC,
- an e-Commerce website for a large company in the telecommunications industry, hereafter labeled LTC,
- a social networking website, hereafter labeled as SNH.

Web access logs were used by all these five websites to keep track of local website activities, which also contain related access failure information as captured in the response code [1]. Except for SNH, where we do not have access to its Web logs, we extracted defect information from these Web access logs. However, for SNH, we do have access to the reported defects from the defect tracking tool used by the host. For LTC, we also have access to the defect database where functional defects from development activities were recorded. Both these data and the Web access log data are used in an analysis to produce our consolidated risk profile for LTC, as described later.

The SMU/SEAS website can be a representative sample of an academic website, sharing many common characteristics of websites for such institutions. It utilizes the Apache Web Server that is commonly used in similar websites. The server log data covering 26 consecutive days in 1999 were used here.

The KDE website is our representative sample of the large number of open source projects who rely on their websites to provide project information, to support online download of released documentation and software, and to facilitate communication and cooperation among the members of large open groups

consisting of software engineers from all over the world. KDE functions as a network transparent contemporary desktop environment for UNIX workstations. Unlike academic websites, such websites are more volatile, with numerous changes continuously committed to provide the developers and users with the most up-to-date version of the produce and related information. With the help of KDE project personnel, we obtained Web access logs from the KDE project for our research on defect analysis for open source projects [24]. Web access log data covering 22 days in 2003 were used here, with significantly larger amount of data than that for SMU/SEAS.

The SCC website is our representative sample of small e-Commerce websites with limited functionality, typically used for small companies as an online catalog showroom to provide information about their products. It uses HP Proliant as the Web server running Redhat ES. In addition to the static pages, the contents of most requested pages are dynamically generated on-the-fly using PHP, Javascript and Perl scripts based on the open source software package "Gallery" with modifications. We used the access log data covering 31 days in 2006 in this study. Because of its small size and combination of both static and dynamic elements, we started our exploratory investigation of risk-prioritized testing with this website as a follow-up to our Web defect classification and analysis [21].

The LTC website is our representative sample of advanced e-Commerce websites often deployed by large companies with a strong Web presence. It is an online ordering application for a large telecommunications company, providing a wide range of services, including: browse available telecom services, manage customer account information, submit inquiries, support for online orders, and request repair for current services. The traffic volume is by far the largest among the websites we included in this study, handling several million requests a day. IIS 6.0 (Microsoft Internet Information Server) was used, supported by various software modules consisting of hundreds of thousands of lines of code developed using Microsoft technologies such as ASP, VB scripts, and C++. In addition to the Web access logs from 2007 used in this study, we also used data from the repository for development defects to produce our consolidated risk profile [20]. Data were also collected in early 2014 that covers the last 4 releases in 2013, with a refined risk prioritization method [18].

The SNH website is our representative sample of the now popular social networking websites. It provides many of the functions similar to the more popular Facebook. We previously used this website and the unstructured data from its defect tracking tool to automatically generate defect classification information based on natural language understanding and learning algorithms [23]. It was also used in our study of usability problem identification and improvement [22]. While without the access to its Web logs, we still consider it an important type of websites to include here. Data from its defect tracking tool collected over 7 months in 2009-2010 were used here.

We have also applied similar ideas to Cloud computing hosted as Web services, including proactive termination of executions predicted to fail based on the Google cluster dataset collected in May 2011 [25], and reliability and usability studies for Google Maps APIs and YouTube APIs based on analyzing selected data sets about usage, defect, and online discussions from 2015 to 2022 [26], [27]. However, we did not perform defect discovery profile comparison due to the huge data size for Google Clusters, and due to the lack of exact execution log data for Google Maps APIs and YouTube APIs. On the positive side, these extensions of the work reported in this study showed the general applicability of similar ideas beyond traditional Web applications to Cloud computing and APIs.

3. Risk-Prioritized Testing

Based on defect analysis we performed on the Web log data or other development defect repositories, risk-prioritized testing can be performed, as described below.

3.1. Web Log Analysis for Risk-Prioritized Testing

To keep a website operational, various Web logs are commonly used to monitor Web accesses and usage and to track the status of related problems. These logs can be analyzed to support our risk-prioritized Web testing without incurring too much additional cost. The commonly used Web logs by Web servers are: Web *access logs*, which record information about individual Web accesses, and Web *error logs*, which record related problems. Some basic problem information can also be obtained from Web access logs directly by examining the response code recorded for each access.

In our approach, we use *defect density* to rank different areas for test prioritization. An specific area is defined by a specific set of conditions or some common characteristics. For example, all files of a common type or with a common extension (which also accounts for slight variations such as ".html", ".HTM", etc.) can be defined as an area. The defect density for a given area is defined to be the ratio of number of unique defects over the number of unique files. With the Web server access logs, the (access) failures can be easily mapped to internal faults by identifying unique causes to these failures, or the individual missing or malfunctioning files. So, for a given area or a subset of files our defect density is the ratio of the number of problematic files over the total number of files in the same subset.

To select among the massive number of different ways to define an area or a group of files, a systematic scheme is needed. This study is based on our previous work on Web defect classification [21] by adapting the Orthogonal Defect Classification (ODC) framework for traditional software systems [28]. Among the Web defect attributes, many are based on the "failure" view, i.e., the type of problems observed and recorded in Web access logs. However, the defect density metric we defined and used for our test prioritization is the "fault" view, i.e., the number of unique internal defects, or faults, over the number of files. For this study, we used the "file type", "directory level", and "owner" type that can be uniquely associated with both individual files and individual faults. We exclude the other defect attributes, such as "access time", "referrer type", and "agent type" that are more closely linked to individual accesses instead of individual files.

With the use of this systematic defect classification and analysis scheme, areas or file subsets can be defined by files of a specific type, at a specific directory level, or owned by a specific type of owners. Among these areas, the ones associated with high defect density values are identified as high risk areas. The ranking order is in accordance to the defect density values, to give us our risk profile, or the prioritized list of testing areas. Then testing would proceed by following this ranking order, as will be the related defect fixing order as well. Table 1 is such an example where the high risk areas are identified and ranked by their defect density for the SCC website.

Table 1. Prioritized Risk Areas for SCC

Risk area	Defect density
Level 4	100%
Icon files	100%
Txt files	100%
Other files	84%
Style files	25%
Static page files	5%
Graph files	2%
Dynamic page files	1%

3.2. Consolidated Defect Prioritization for Risk-Prioritized Testing

For e-Commerce and other business applications deployed over the Web, more disciplined development

methodologies are used. All the defects from the development and maintenance processes, discovered through testing, inspection, other quality assurance activities, and normal customer usage, need to be formally tracked, logged, and resolved. Data from such defect tracking tools or defect repositories provide us with additional valuable input for problem analysis and focused quality improvement. The recorded defect information for the development of software used to support the operations of the LTC website include the following fields: 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.

In our study, defects recorded in such product development defect repositories are classified as *functional defects*, which represent incorrect or missing implementations of required functions expected by target customers and users. Such defects may only be detected in the development or system maintenance processes but not through analyzing Web server logs, because a missing function will not be accessed, and a incorrect function can still be accessed with or without an access problem but with wrong results. Table 2 gives the distribution of such functional faults for the LTC website. The corresponding HTTP categories for these defect are also given based on the hypothetical situation of a website with such defects left untouched. The top three categories, covering defects that would not be detectable through Web access log analysis because they would result in normal response code, represent 76.50% of the total defects. Only 10.63% of the problems from the defect repository can also be found in the server Web logs as missing files with response code 404. This insignificant overlap between the two data sources indicates that we need to study both for risk-prioritized testing of such Web applications.

Table 2. Functional Faults Discovered for LTC

Fault class	HTTP categories	% of total
Service/system interfaces	200/300	33.13%
Graphical user interfaces	200/300	22.89%
Code logic, computation and algorithm	200/300	20.48%
Missing files	404	10.63%
Missing links	200/300	9.04%
Cache	200/300	1.20%
All other fault classes	200/300/400/500	2.63%
All	200/300/400/500	100.00%

To build a consolidated risk ranking, we need to merge the functional defect data from development defect repository and data from Web access logs. The defect repository data can be ranked by their share among the total number of defect, give us a partial risk profile. Notice here that we used defect share, a slight variation of defect density we defined earlier, because defect density per file would be less meaningful for files associated with source code. Source code files are not access each as a unit as is the case for Web files. In fact, defect density for source code is typically defined as the number of defect over the number of lines of code (LOC), which would be incompatible with defect density per Web file we defined earlier.

As described earlier in this section, access failures from Web access logs can be easily mapped to missing file faults by identifying unique causes to such failures. Then, these missing file faults can be directly merged with the functional faults derived above to build the collective risk profile. Again, fault share is used in this ranking. We need to re-normalize the fault share, considering all the fault classes after removing duplicate entries. Table 3 shows the comprehensive risk profile for the LTC website, which gives the consolidated fault ranking by category.

Table 3. Top Classes of Faults (Collective Risk Profile) for LTC

Fault class	% of total
Missing files – HTTP 404 failures	33.64%
Service/system interfaces	25.34%
Graphical user interfaces	17.50%
Code logic, computation and algorithm	15.67%

3.3. Implementation

Risk-prioritized testing can be performed in two stages:

- *Stage 1:* Risk identification, which will produce a prioritized list of risk areas based on analyzing some existing data. This data set is typically called the training data, which is used to provide training for the risk-prioritized testing technique.
- *Stage 2:* Performing testing based on the prioritized list of risk areas above, typically in descending order according to the list. The data used for this comparison will be called testing data. Not to be confused with software testing, the term testing used here is for hypothesis testing.

Stage 1 gets started once some data are accumulated and deemed to be adequate based on some threshold defined on the chronological sequence of the data. A given threshold on the amount of data or external logical event will get Stage 1 started, with its results used in Stage 2 to actually perform risk-prioritized testing.

With the training data, a risk-based prioritization list, or our *risk profile*, will be produced using our technique described above. In the actual implementation of the risk-prioritized testing strategy, this risk profile will be used to determine the execution sequence of different areas of testing, i.e., the test cases are selected and executed in strict order determined by the risk profile.

4. Empirical Evaluation

In this section, we empirically evaluate the effectiveness of different testing techniques by comparing their corresponding defect discovery profiles.

4.1. From Reliability Growth to Defect Discovery Profiles

In software reliability engineering (SRE), failure observations from testing are typically plotted against time and fitted to various software reliability growth models (SRGMs) [2, 3]. The reliability *growth* due to defect removal by software developers as follow-up actions to testing can be visualized in such plots. The horizontal axis represents the elapse time or effort expended. Because software or Web application failures are triggered by active usage in either the normal operational or the testing environment, usage time measured by the execution time, number of test runs, or workload handled are typically preferred over raw calendar time or wall-clock time [3, 29]. The vertical axis represents the failure observations, using either the raw failure counts for each period or the cumulative failure data.

When cumulative failure data over time are used, the reliability growth is seen as the gradual flattening of the failure curve as we move forward with time from left to right in the horizontal axis. Fig. 1 presents an illustrative example of such a plot. A curve with a sharper upward bend toward the upper-left corner would indicate more reliability growth, because it represents more failure observations near the beginning, typically accompanied by more fault removal because every effort is made to address the issues from testing in mature software development organizations. This initial quickened pace of defect discoveries will be followed by more pronounced reduction of failure rate (the flatter slope of the curve) towards the end because of the significantly reduced number of defects in the software due to defect fixing activities taking

place immediately after their initial detection through testing earlier.

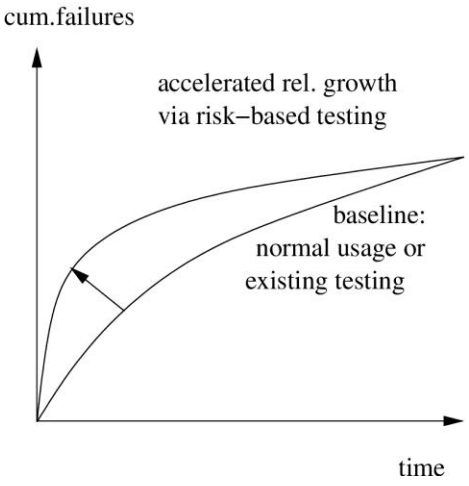

Fig. 1. Accelerated reliability growth or defect discovery over time.

Because of the internal testing and defect fixing will be applied to source components (modules, files, units, or individual lines of code) where such faults are observed, we will use the accumulation of such components tested as the horizontal axis. For Web applications, the natural unit is the individual files. This choice is also in agreement with the defect density metric, number of defects per file, we used for our risk-prioritized testing described earlier.

Therefore, we can plot defect (fault) discovery profiles over accessed Web files to examine the accelerated defect discovery. The shape of the defect discovery profile has important implications to resource allocation, scheduling, and testing efficiency. Similar to plots such as in Fig. 1, we would prefer curves that bend toward the upper-left corner. There are several characteristics to such desirable profiles:

- Discovering more defects earlier will give development teams more time and adequate resources to fix the problems and testing teams more time to retest to verify that the problems have indeed been resolved before product release.
- In the competitive software market, time-to-market pressure often dictates the product release time. Under such compressed schedule, a defect discovery profile that bends towards upper-left corner would allow the maximal number of defect to be discovered, worked on, and resolved before (sometimes premature) product release.

When risk identification results and risk-prioritized testing are used, we expect that the curve would bend more towards the upper-left corner, with faster problem detection and correction in the early part and a flatter tail. In fact, this would be the hypothesis that we attempt to test in this section: a favorable defect profile of risk-prioritized testing as compared to other testing strategies.

4.2. Experimental Design

Based on the above discussion, we need to produce the defect discovery profile for risk-prioritized testing and compare it with that for other types of testing. Similar to the implementation of our risk-prioritized testing strategy discussed in the previous section, for post-mortem comparison, we can divide the data into a training set and a testing set at approximately the halfway point. Other cutoff points in accordance to other external reasons, such as significant change occurring around a certain point, can also be used.

The results of our risk-prioritized testing can be simulated using the testing data by sorting the data according to the risk profile we produced from the training data above, accompanied by the corresponding

defect discoveries. Consequently, the resulting defect discovery profile will resemble the one in which risk-prioritized testing is actually implemented for Web application testing.

Similar to the simulated implementation results for risk-prioritized testing, other testing strategies can be simulated or hypothetically performed on the testing data by sorting the data according to the components or files accessed and the corresponding problem observations. In this case, for various coverage-based testing strategies, only the testing data set will be used to produce the corresponding defect discovery profiles. Based on different coverage criteria, the testing data are sorted in the following orders meaningful to the Web domain:

- dictionary (or alphabetical) order,
- reverse dictionary order,
- ascending directory levels,
- descending directory levels,
- random directory levels.

These coverage orders correspond to some of the most commonly used coverage testing methodologies used in the Web environment. Dictionary order is to test or cover all the files in a website by the alphabetical order of their full file names. Because the full file names include the file path information, some structural information is used to cover different areas in a systematic manner: all the Web files in the same sub-directory will be tested before testing for other sub-directories start. Therefore, dictionary order testing simulates the test scenario where the website is tested one sub-site or sub-directory at a time, in alphabetical order, until complete directory coverage is achieved. Reverse dictionary order test the website in the exact opposite order, but also achieve complete directory coverage.

In general, Web design guidelines for contents organization recommend that important files be placed at as close to the access point as possible, which usually means that they should not be buried deep in the directory structure [30]. Files close to the top levels of the directory structure are generally more important to and frequently accessed by a large pool of users. Therefore, it makes sense to conduct coverage-based testing in the order of ascending directory levels. For comparison and contrast, we also include testing in descending directory order, and random directory order. In the case a given list is maintained regarding the directory level order to test, we could simulate that as well.

4.3. Results for Websites Using only Web Log Data

We first experimented with the SCC website, because of its relatively smaller size that allowed us to explore the various issues and to fine tune the related software tools we developed to support our risk-prioritized testing and comparative analysis. Then, we replicated the experiment in the two other websites, SMU/SEAS and KDE in an attempt to scale-up the evaluation study using significantly larger websites with much more users and user hits. Fig. 2 shows the defect discovery profile of our risk-prioritized testing for SMU/SEAS and compares it to that for other common coverage schemes: random order, usage order, dictionary order, reverse dictionary order, sorted directory level order, and random level order.

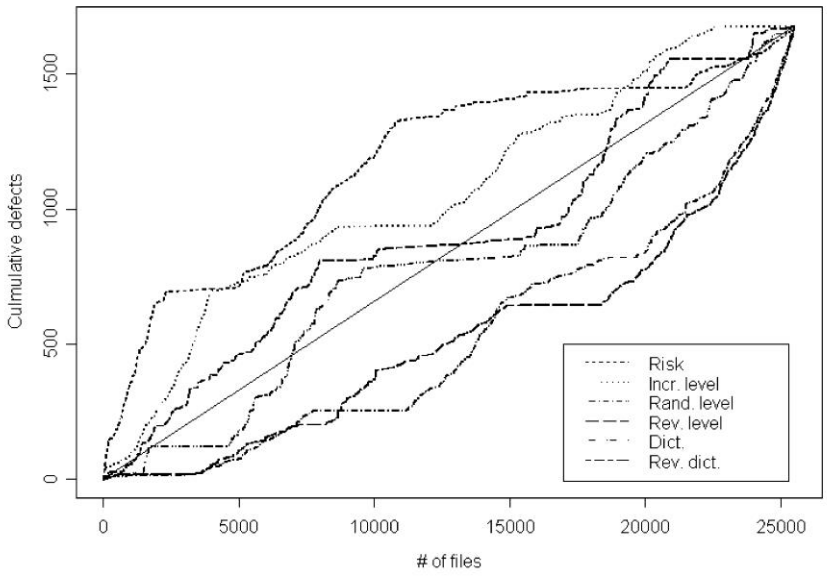

Fig. 2. Defect discovery profiles for SMU/SEAS over the number of files tested/accessed.

As seen in Fig. 2, the risk-prioritized testing compares favorably to other coverage-based testing, leading to discovering more defect upfront. Similar patterns were also observed for SCC and KDE. There is a particularly significant difference early in the testing sequence: Our risk-prioritized testing can lead to the discovery of 80-90% of all the defects (faults or unique failures) in the first 1/3 files tested or accessed in Fig. 2. This effect is even more pronounced in SCC, where more than 600 out of about 800 defects can be detected by testing (or attempting to access) and examining a few hundred files out of nearly 9000 ones. This is because of the high concentration of defects in a small number of files as predicted from the training set in Table 1.

4.4. Results for Websites Using Consolidated Defect Data

As described in the previous section, we have different data sources for e-Commerce websites, which can be consolidated to produce our comprehensive risk prioritization based on both Web log data and development defect repository data. We performed the same defect discovery profile comparison for LTC and presents our results in Fig. 3.

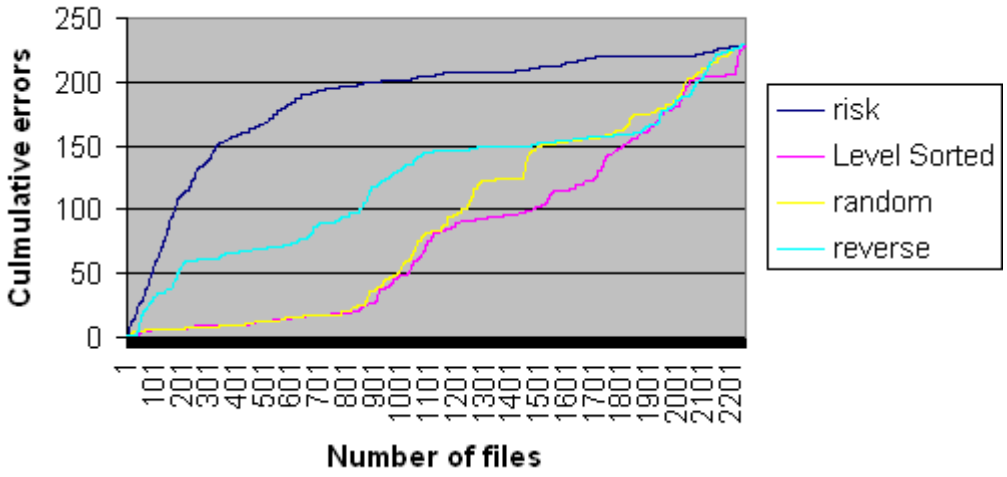

Fig. 3. Defect discovery profiles for LTC over the number of files tested/accessed.

Unlike the informational websites in the previous group, the LTC website is more tightly integrated into the lower level business logic, associated software component, and the underlying database. The comprehensive risk profile or the consolidated defect prioritization is used. In addition, some of the coverage schemes we used before would be less meaningful for this particular setting. For example, order of names (dictionary order or its reverse) for different types of files (e.g., comparing program files for underlying components and for the dynamic Web with HTML and other web documents) would not be very meaningful, and thus omitted from our comparison.

Fig. 3 offers a strong evidence about the scalability and general validity of our hypothesis that risk-prioritized testing can accelerate defect discovery. The defect discovery profile for risk-prioritized testing consistently outperformed all other testing by a wide margin. In particular, it can lead to the discovery of more than 70% of the defects with less than a third of the files tested or examined. At the same cutoff for the files examined, all other strategies can only reveal a third of the defects or less.

For the SNH website, we only have access to the defects from its defect tracking tool but not the Web logs. This data availability issue prevented us from using the defect density defined in Section 3, because of the missing count of files in different groups. However, we could produce a rough profile, ordered by the defect prioritization areas but not with exact scale for the horizontal axis. Table 4 gives the risk-based prioritization for SNH based on the training set as well as the actual defect distribution for the testing set. Notice that we used defect share instead of defect density in this table, giving only the share (or percentage) of total defects by each defect group. The order is roughly the same over different areas in the two sets. If risk-prioritized testing is used, those identified risk areas will be targeted first, leading to the discovery of similar shares of defects in the testing set. This would compare favorably to most other orders because in the testing set, a reshuffled order will lead to less fault discoveries upfront unless the first few "high-risk" areas, which contain more actual defects in the testing set too, were targeted first.

Table 4. Prioritized Risk Areas for SNH

Risk area	Defect share (% of total)	
	Training	Testing
Function failure	81.4%	70.5%
Function workaround	11.0%	4.8%
User interface	6.1%	7.6%
Crash	1.5%	2.9%
Minor requirement	0%	3.81%
None	0%	10.5%

5. Conclusions and Perspectives

As demonstrated through the replicated experiments over five websites from diverse Web application domains described in this paper, risk-prioritized testing is an effective and efficient way to help website owners and maintenance teams to accelerate their defect discovery and reliability improvement. We used the training data from Web logs and development defect repositories to produce our risk profiles or our prioritized testing areas. For the empirical evaluation, we did have tight experimental control in simulating expected behavior when different testing strategies are used in the testing data. The diversity of the websites used, covering academic, open-source project, small business catalog showroom, large e-Commerce application, and social networking Web application domains, gives us high confidence to the general validity of our conclusions. Extensions of this work to Cloud computing and APIs showed the general applicability of similar ideas to a wider variety of application domains [25–27].

We also noticed some limitations to our study due to its limited scope and other constraints. For volatile websites, prediction accuracy of the risk areas may decrease because of the significant difference expected between the training and the testing sets, leading to inappropriate risk prioritization. We need to fine-tune our method to use only the most recent and most relevant data for test prioritization. In addition, different prioritization schemes will yield different, not always consistent, results: A strategy aimed at maximizing reliability growth may not produce the same effect on maximized defect discovery for a given time period.

As a follow-up to this study, we will explore other innovative use of defect, usage, and product internal information in risk profiling in an attempt to achieve even more accelerated defect discovery and reliability improvement. For example, the testing order by risk profile areas in this study can be refined using finer granularity defect density metrics, such as normalized over the number of hyperlinks, and dynamically adjusted to always focus on the area with the highest current defect density in a fine-grain, adaptive risk-prioritized testing strategy.

In addition, risk-prioritized testing can be compared to other variations of testing not covered in this study, including many existing coverage orders and coverage hierarchies, such as equivalent class partition testing, control flow and data flow testing, boundary testing, etc., [5]. To make the comparison meaningful, our risk prioritization scheme might need to be adjusted to produce a risk ranking that utilize the specific coverage related definitions as well. For example, for comparison to partition coverage testing, our risk ranking would be based on the defect density for the same set of partitions. In fact, the comparison in this paper can be interpreted as comparing our risk-prioritized testing to a specialized partition testing where the partitions are defined by file names or file location in the directory system.

The empirical evaluation can be expanded to include entities other than defect discovery profiles. Reliability profiles from the operational view can be directly used to compare different testing strategies, as we did in [1, 29], plotting cumulative failures against cumulative usage time. In doing this, the prioritization scheme will be based on failure intensity for different subclasses of usage scenarios instead of the defect (or fault) density per file used in this paper. To do this for e-Commerce websites, we need to evaluate the potential impact of functional faults based on defect severity and likely usage scenarios. When combined with defect data from web server logs, it gives us a collective failure view [18, 20], and provides data input for our focused testing and reliability improvement.

We also plan to work on practical implementation and tool support to deploy our approach in "live" environments to reap the benefit of accelerated defect discovery for many website owners, operators, and service providers to better serve their customers and users. Effort comparison can also be carried out, comparing the overhead involved in implementing our risk-prioritized testing against that for traditional coverage-based testing or usage-based testing. The overhead of our approach should be justified by the accelerated defect discovery against coverage- or usage-based testing.

Conflict of Interest

The author declares no conflict of interest.

Funding

This research was supported in part by the NSF Grant #1126747, Raytheon and NSF Net-Centric I/UCRC.

References

[1] Kallepalli, C., & Tian, J. (2001). Measuring and modeling usage and reliability for statistical Web testing. *IEEE Trans. on Software Engineering*, 27(11), 1023–1036.

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

[3] Musa, J. D. (1998). *Software Reliability Engineering*. McGraw-Hill, New York.
[4] Offutt, J. (2002). Quality attributes of Web applications. *IEEE Software*, *19(2)*, 25–32.
[5] Beizer, B. (1990). *Software Testing Techniques.* International Thomson Computer Press, Boston, MA.
[6] Chen, M. H., Lyu, M. R. & Wong, W. E. (2001). Effect of code coverage on software reliability measurement. *IEEE Trans. on Reliability*, *50(2)*, 165–170.
[7] Malaiya, Y. K., Li, M. N., Bieman, J. M., & Karcich, R. (2002). Software reliability growth with test coverage. *IEEE Trans. on Reliability*, *51(4)*, 420–426.
[8] Frankl, P. G., Hamlet, R. G., Littlewood, B., & Strigini L. (1998). Evaluating testing methods by delivered reliability. *IEEE Trans. on Software Engineering*, *24(8)*, 586–601.
[9] Mills, H. D. (1972). On the statistical validation of computer programs. Technical Report 72-6015, IBM Federal Syst. Div.
[10] Whittaker, J. A., & Thomason, M. G. (1994). A Markov chain model for statistical software testing. *IEEE Trans. on Software Engineering*, *20(10)*, 812–824.
[11] Hamlet, D., & Taylor, R. (1990). Partition testing does not inspire confidence. *IEEE Trans. on Software Engineering*, *16(12)*, 1402–1411.
[12] Ntafos, S. C. (2001). On comparisons of random, partition, and proportional partition testing. *IEEE Trans. on Software Engineering*, *27(10)*, 949–960.
[13] Voas, J. M. (2000). Will the real operational profile please stand up? *IEEE Software*, *17(2)*, 87–89.
[14] Elbaum, S., Malishevsky, A.G., & Rothermel, G. (2002). Test case prioritization: A family of empirical studies. *IEEE Trans. on Software Engineering*, *28(2)*, 159–182.
[15] Frankl, P. G., & Weyuker E. J. (2000). Testing software to detect and reduce risk. *Journal of Systems and Software*, *53(3)*, 275–286.
[16] Cukic, B. (2005). The virtues of assessing software reliability early. *IEEE Software*, *22(3)*, 50–53.
[17] Srikanth, H., & Kan, S. H. (2008). Impact of defect backlog on product release and quality. *Software Quality Professional*, *10(3)*, 27–35.
[18] Jaffal, W., & Tian, J. (2014). Practical risk-based technique to improve reliability for incremental Web application development. *Proceedings of the 27th Int. Conf. on Computer Applications in Industry and Engineering.*
[19] Karami, G., & Tian, J. (2018). Maintaining accurate web usage models using updates from activity diagrams. *Information and Software Technology*, *96*, 68–77.
[20] Li, Z., Alaeddine, N., & J. Tian, J. (2010). Multi-faceted quality and defect measurement for web software and source contents. *Journal of Systems and Software*, *83(1*), 18–28.
[21] Ma, L., & Tian, J. (2007). Web error classification and analysis for reliability improvement. *Journal of Systems and Software*, *80(6)*, 795–804.
[22] Geng, R. & Tian, J. (2015). Improving web navigation usability by comparing actual and anticipated usage. *IEEE Trans. on Human-Machine Systems*, *45(1)*, 84-94.
[23] Huang, L., Ng, V., Persing, I., Chen, M., Li, Z., Geng, R., & Tian, J. (2015). AutoODC: Automated generation of orthogonal defect classifications. *Automated Software Engineering*, *22(1)*, 3-46,
[24] Koru, A. G., & Tian, J. (2005). Comparing high-change modules and modules with the highest measurement values in two large scale open-source products. *IEEE Trans. on Software Engineering*, *31(8)*, 754–769.
[25] Tian, Y., Tian, J. & Li, N. (2020). Cloud reliability and efficiency improvement via failure risk based proactive actions. *Journal of Systems and Software*, *163*.
[26] Bokhary, A., & Tian, J. (2017). Cloud service reliability assessment and prediction based on defect characterization and usage estimation. *Int. J. of Computer and Their Applications*, *24(2)*, 63–70.

[27] Tian, J., Alanazy, S., Bokhary, A., Alharthi, S., & Ghanem, S. (2022). Measuring influencing factors of API usability. *Proceedings of the 9th Annual Conf. on Computational Science & Computational Intelligence (CSCI'22)*, Las Vegas, NV.

[28] Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., & Wong, M.-Y. (1992). Orthogonal defect classification—A concept for in-process measurements. *IEEE Trans. on Software Engineering*, *18(11)*, 943–956.

[29] Tian, J., Rudraraju, S., & Li, Z. (2004). Evaluating Web software reliability based on workload and failure data extracted from server logs. *IEEE Trans. on Software Engineering*, *30(11)*, 754–769.

[30] Shneiderman, B., Plaisant, C., Cohen, M. S., Jacobs, S. M., & Elmqvist, N. (2017). *Designing the User Interface: Strategies for Effective Human-Computer Interaction, 6th Edition*. Addison-Wesley, Reading, Massachusetts.