Reliability Assessment of Mass-Market Software: Insights from Windows Vista®

Paul Luo Li, Mingtian Ni, Song Xue, Joseph P. Mullally, Mario Garzia, Mujtaba Khambatti

Microsoft Windows Reliability Team
One Microsoft Way
Redmond, WA USA
Paul.Li@Microsoft.com

Abstract

Assessing the reliability of mass-market software (MMS), such as the Windows® operating system, presents many challenges. In this paper, we share insights gained from the Windows Vista® and Windows Vista® SP1 operating systems. First, we find that the automated reliability monitoring approach, which periodically reports reliability status, provides higher quality data and requires less effort compared to other approaches available today. We describe one instance in detail: the Windows Reliability Analysis Component, and illustrate its advantages using data from Windows Vista. Second, we show the need to account for usage scenarios during reliability assessments. For pre-release versions of Windows Vista and Vista SP1, usage scenarios differ by 2-4X for Microsoft internal and external samples; corresponding reliability assessments differ by 2-3X. Our results help motivate and guide further research in reliability assessment.

1. Introduction

Mass-market software (MMS) is prevalent and important today. In the past, reliability research has focused on custom-built software systems, e.g. NASA space shuttle software [22] and AT&T telephone switching software [19]. However, MMS such as Windows are essential to the lives of hundreds of millions of people today. US Navy navigation systems [9] and ABB power plant IT systems [1] run on top of Windows. Unlike custom-built software systems, MMS are intended to be used by a variety of users in different usage scenarios.

Ensuring that MMS meet the reliability expectations of customers is important. Reliability improvements can mean billions of dollars in revenue and in saving from repairs for software producers [20] and increased utility for users.

Lack of reliability feedback data hinders the ability of software producers to meet the reliability expectation of customers. By reliability feedback data, we mean information on failures that users encounter during real-world usage. Experiences at several organizations [18] have shown that software producers need reliability feedback data to assess the reliability of their products and to identify and address failures, in order to meet the reliability expectations of customers.

Various approaches are used in industry today to collect reliability feedback data. In Section III, we organize approaches into four classes and compare them using seven criteria that are important to software producers. The question we address is:

How do approaches used to collect reliability feedback data differ?

We find that the automated reliability monitoring approach provides higher quality data (accurate, correct, comprehensive, normalized, and actionable) and requires less effort (effort by software producers to collect the data and effort by users to provide the data) compared to other approaches available today. In section IV, we describe one example, Windows Reliability Analysis Component (RAC), in detail and illustrate its advantages using data from Windows Vista.

Even with good reliability feedback data, assessing the reliability of MMS is difficult; one issue is misleading assessments resulting from diverse usage scenarios. Musa theorizes in [19] that the reliability of a software system can differ depending on the scenarios in which it is used. This is a problem for MMS, which can be used in many different scenarios. If scenarios used to assess reliability are different from actual customer usage scenarios, then misleading assessments can result. This may lead software producers to take sub-optimal actions pre-release. Recent research efforts have focused on this issue, e.g. [7], [8], and [15]; however, there is little quantitative data on usage scenarios differences and corresponding reliability differences. The question we address is:

How much can usage scenarios differ and what are the corresponding reliability differences?

In Section V, we use data from pre-release versions of Windows Vista and Vista SP1 to show that usage differences between Microsoft internal and external samples can be 2.2-3.7X (measured using applications executed) and the applications that experienced failures differ by 2.8-3.4X. The corresponding reliability differences are 1.7-3.1X (measured using rate of failures).

2. Prior work / Motivation

Ensuring that users have good reliability experiences is important to producers of mass-market software (MMS) as discussed by Buckley and Chillarege [4] and Chulani et al. [6] of IBM. Consequently, software producers need information on failures that impact users' reliability experiences. Other kinds of reliability feedback data, such as user satisfaction survey-data and testing data, are not sufficient. User satisfaction survey-data can provide information on users' reliability experience but do not identify specific failures that software producers can address. Testing data can identify specific failures but do not show impact on users' reliability experience.

By failures we mean disruptions to the user, such as application crashes, application hangs, and OS crashes. In contrast, most prior work equate failures to bugs in the code,

e.g. change requests [4] and code check-ins with references to bug numbers [17]. These do not necessarily reflect users' reliability experiences. More than one failure can result from a single bug in the code, some bugs may not manifest as failures noticed by users, and some users may not report failures [16]. An underlying issue is the approaches used to collect the reliability feedback data. These approaches, typically call-centres and web-based bug reporting tools, depend on users to self-report failures; however, these approaches have many drawbacks as discussed by prior work, e.g. Murphy in [18], Ostrand et al. in [21] and Sharma and Jalote in [23]. We discuss the drawbacks in detail in Section III.

Recently, improved approaches that overcome drawbacks of prior approaches have been introduced, notably automated per-incidence reporting, such as Windows Error Reporting [26]. However, no prior work has examined the approaches in terms of their ability to help software producers meet the reliability expectations of customers.

Results from such an evaluation can help researchers to choose better data sources for their studies and can help software producers to adopt better approaches for collecting reliability feedback data.

One important use of reliability feedback data is to estimate users' reliability experience as discussed in many prior studies, e.g. Musa [19] and Garzia et al. [11]. A common assumption is that usage scenarios used to assess reliability matches customers' usage scenarios. Capturing and leveraging usage scenarios in reliability evaluations for web-based applications have shown promising results as reported by e.g. Goseva-Popstojanova et al. [12] and Weyns and Martin [24]. However, measuring and accounting for differences in usage scenarios, e.g. Musa's work to define and utilize usage profiles [19], have proven to be difficult for applications and operating systems due to complex environments and interactions. This has motivated research to profile deployed software and to account for differences, e.g. Diep et al. in [8], Andrzejewski et al. [2], Clause and Orso in [7] , and Jeske and Akber-Qureshi in [13].

However, no prior work has quantified differences between usage scenarios and corresponding impact on reliability assessments. Such findings can help motivate on-going research.

3. Evaluation of approaches used to collect reliability feedback data

In this section, we present four classes of approaches, seven criteria used to compare approaches, and then the comparison.

We examine four classes of approaches used to collect reliability feedback data for mass-market software (MMS). We focus on the most commonly-used approaches. Other specialized approaches exist; however, many do not apply to MMS. For example, Mockus et al. describe a feedback system for embedded telephone switching systems in [17], but it runs on a small set of hardware configurations, operates with a specific set of applications, and is used in known scenarios. We compare classes and not individual instances because of legal and technical considerations; without access to all the instances, we cannot guarantee that our comparisons are correct.

Reliability feedback data include information that identifies the occurrences of a failure, e.g. the faulty application and the errant behaviour, along with information on relevant system and environment states. Software producers typically use reliability feedback data in two ways: 1) evaluate the reliability status of the product 2) identify and address failures.

Our comparison presumes construct validity, i.e. the measured failures actually impact users' reliability experiences. We mean failures to be application crashes, application hangs, and OS crashes; these issues are known to disrupt users. Nonetheless, other reliability issues that impact users' reliability experiences exist. Consequently, software produces need to periodically validate their data and make appropriate updates. This validation process may or may not use the approaches discussed in this paper. For example, the validation process may include detailed questionnaires to understand the reliability experiences of users. The approaches we discuss are used in everyday data collection processes after construct validity of the data has been verified.

3.1. Four Classes of Reporting Approaches

1. **Interactive user reporting**: For each failure, the user initiates the feedback process by contacting the software producers. A live operator interacts with the user to collect the feedback data by filling out a predefined form. A developer may contact the user later to ask for additional information. This is a detailed study to understand users' reliability experiences.

   Examples: call centres, IM centres
2. **Online user reporting**: For each failure, the user initiates the feedback process using a feedback application or through a website. The user provides the feedback data by completing a pre-defined form. A developer may contact the user later to ask for additional information.

   Examples: Bugzilla, BSD's Send-bug
3. **Automated per-incidence reporting**: The system automatically initiates the feedback process when it detects the occurrence of a failure. The system automatically collects the necessary feedback data and submits it to the software producer with user's consent.

   Examples: Apple's Crash Reporter, Google/Mozilla Breakpad, Windows Error Reporting
4. **Automated reliability monitoring**: After initial consent, the system automatically sends periodic reliability status to the software producer. The report contains data on all the failures that has occurred during the period. In addition, the system reports successful executions and execution durations.

   Examples: Window's Reliability Analysis Component

3.2. Evaluation Criteria

The seven evaluation criteria below are important to MMS producers; they have been important for Windows. The lack of any criterion may jeopardize the ability of software producers to meet the reliability expectations of customers.

• **Accurate**: issues reported are failures

In order to maximize the return on reliability improvement efforts, software producers need to avoid issues that are not failures, e.g. issues resulting from user mistakes, which can cost valuable time and resources to investigate. Augustine and Podgurski discuss this issue in [3].

• **Correct**: all data values in reports are correct

Software producers need correct information to address failures. Data entry mistakes or omissions can lead to un-diagnosable or un-reproducible failures. Murphy discusses this issue in [18].

• **Comprehensive**: all failures are reported

Software producers typically need to prioritize failures, which cannot be efficiently done if missing failures. Software producers cannot weight failures appropriately if users are not reporting failures or under-reporting failures. Li et al. discuss this issue in [14].

• **Normalized**: reliability data can be normalized by usage
Software producers typically need to compare reliability between applications, releases, or across time. Increased usage may lead to more failures, which will be indistinguishable from effects of a less reliable product. We discuss this issue in detail in Section IV.

• **Actionable:** data enables developers to fix failures
To address failures, software producers often need in-depth information. This may include information on the system/application state or information on the execution environment. Diep et al. discuss this issue in [8].

• **Minimal effort by software producer:** data is collected at low costs
Reliability information can be costly for software producers to collect, unless they use some automation. Mockus et al. discuss this issue in [17]. This effort is independent of the effort needed by developers/analysts to diagnose and fix a failure.

• **Minimal effort by users**: users are not inconvenienced during the feedback data collection process
A goal of reliability improvement is to improve customer satisfaction. Buckley and Chillarege [4] have shown that repeated user disruptions negatively impact customer satisfaction; consequently, software producers want to minimize user effort during data collection.

3.3. Comparison

Our experience has been that automated reliability monitoring provides higher quality data and requires less effort; table 1 contains details of our evaluation. Interactive user reporting and online user reporting can have accuracy and correctness problems because users provide the feedback data manually, as noted in [5] and [21]. This manual process can also be a burden on users, which can lead to non-comprehensive reporting, as discussed in [17]. Neither approach captures usage so information cannot be normalized, which is important as shown in the next section. Furthermore, since users report failures from their perspective, the failures may not have sufficient information to be actionable by software producers, as discussed in [7]. Interactive user reporting has the additional problem of requiring live operators to operate, which can be expensive. Automated per-incidence reporting approaches improve in each of the areas, as described in Table 1, except for normalization. Since there are no reports when users do not encounter failures or are not using the system, the data cannot be normalized, e.g. with respect to the number of running machines or runtime. Automated reliability monitoring improves by reporting data periodically, including usage information. In the next section, we illustrate this advantage.

Table 1
Comparison of approaches used to collect reliability feedback data from customers

	Accurate	*Correct*	*Comprehensive*	*Normalized*	*Actionable*	*Minimal effort by software producer*	*Minimal effort by users*
Interactive user reporting approaches	*May not be accurate:* Operators may record non-failures [18]	*May not be correct:* User provided information may not be correct [21]	*Not comprehensive:* Users only occasionally report failures [14]	*Not Normalized:* Usage is unknown because users may not report	*May not be actionable:* Users often report symptoms [8]; limited ability to obtain additional information	*Not automated:* Live operators	*Significant effort by users:* Substantial user effort is needed to report failures [17]
Online user reporting approaches	*May not be accurate:* Users often report non-failures [3]	*May not be correct:* User inputted information may not be correct [21]	*Not comprehensive:* Users only occasionally report failures [14]	*Not Normalized:* Usage is unknown because users may not report	*May not be actionable:* Users often report symptoms [8]; limited ability to obtain additional information	*Automated:* Automated mechanism	*Significant effort by users:* Substantial user effort is needed to report failures [17]
Automated per-incidence reporting approaches	*Accurate:* By construction, only failures are reported	*Correct:* Data are generated by the system	*May not be comprehensive:* Only instrumented failures are reported; may not report because requires user consent	*Not Normalized:* Usage is unknown because no reports occur when no failures occur or when system is not used	*Actionable:* Contains information that enables developers to address failures	*Automated:* Automated mechanism	*Minimal effort by users:* Reports are sent automatically
Automated reliability monitoring approaches	*Accurate:* By construction, only failures are reported	*Correct:* Data are generated by the system	*May not be comprehensive:* Only instrumented failures are reported; may not report because requires user consent	*Normalized:* Usage is known because machines report periodically	*Actionable:* Contains information that enables developers to address failures	*Automated:* Automated mechanism	*Minimal effort by users:* Reports are sent automatically

4. Windows' Reliability Analysis Component

The Windows Reliability Analysis Component (RAC) is a new feature in Windows Vista designed to provide accurate and timely reliability feedback data. RAC collects data from users opted into Windows Customer Experience Improvement Program (CEIP) [25]. RAC is light-weight, secure, and user-privacy compliant; ensuring user trust is a key consideration in the design. RAC runs in the background as a low priority process and gathers data using the event log and system calls. RAC periodically sends data to Microsoft via a secured connection. RAC currently tracks hundreds of thousands of real world Windows Vista machines. Even though the data may be biased because they are only from users opted-in to CEIP, with a large pool and wide variety of users, we believe the data is representative.

RAC follows an automated reliability monitoring approach. The reliability feedback data are generated by the OS; in general, all issues are actual failures experienced by the user (failures can be experienced more than once) and the data are correct. Except for edge cases, e.g. hard OS hangs, users with no internet connectivity, and users not opted into CEIP, all failures are recorded; thus, the data are mostly comprehensive. Users are opted-in by default for all pre-release Windows

releases; users are opted-out by default post-release. Machines periodically report status, so the data can be normalized. RAC is automated and does not disrupt users during reporting. RAC collects problem identification numbers, which can be used by developers to address failures.

4.1. Advantages

The most distinct characteristic of RAC is continuous monitoring rather than per-instance reporting. RAC tracks system and application state changes, which enables normalization with respect to usage, such as the number of running machines, application runtime, and application execution count. *The usage information is the key to normalizing the reliability feedback data for tracking and comparison.* The Windows Reliability Team uses the rate of failures (failure count per running machine) to track OS reliability and the failure exit ratio (failure count divided by application execution count) and the failure frequency (failure count divided by application runtime) to track application reliability.

The advantages of failure rates are clear; without normalization (i.e. knowing the number of running machines) comparisons cannot be made across releases and over time because the number of machines reporting data can change and is usually unknown for MMS. These data have helped Windows to evaluate the reliability release readiness of Windows Vista and Vista SP1.

Normalized metrics also help Windows to prioritize quality assurance efforts. We illustrate the benefits using two examples from Windows Vista. First, we examine two media players. In Figure 1 we see that media player A has ~4.3X the failures of media player B. Using per-incidence reporting, we may have concluded that media player A is less reliable. However, with RAC usage data we observe that media player A is executed ~27.9X times more compared to media player B. Media player A actually has superior reliability because media player B has a failure exit ratio that is ~6.4X greater than that of media player A.

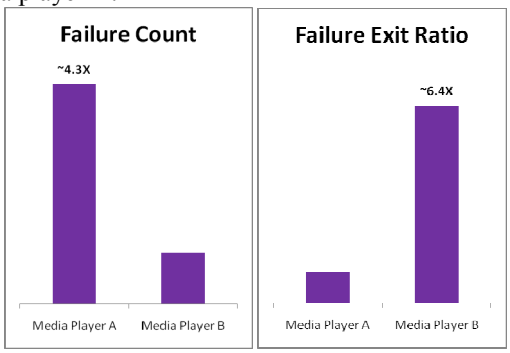

Figure 1. Failure count and failure exit ratio for two media players

Second, we examine two instant messaging applications. In Figure 2, we see that instant messenger A has a higher failure count, ~3.5X more, and a higher failure exit ratio, ~4.2X more, than Messenger B. We may have concluded that instant messenger A is less reliable. However, instant messaging applications tend to be persistent, i.e. users tend to run instant messaging applications until experiencing a failure. Consequently, to gauge reliability, we need to normalize by application runtime. In Figure 3 we observe that instant messenger A appears to be more reliable as a function of runtime; instant messenger B has a failure frequency that is ~2.4X higher. Users have been launching and then exiting instant messenger B before it hits a failure.

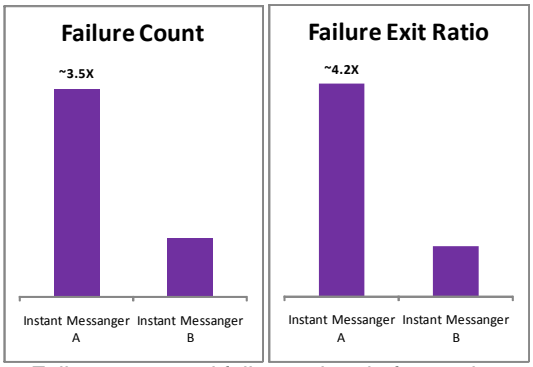

Figure 2. Failure count and failure exit ratio for two instant messaging applications

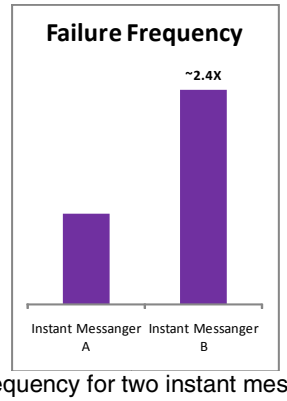

Figure 3. Failure frequency for two instant messaging applications

In addition to normalization, RAC also helps developers by providing contextual information. For example, RAC provides the operating history of the machine, uch as applications and drivers recently installed or updated and other failures experienced. Developers have benefited from such contextual information in triaging and debugging of failures.

5. Problems with scenarios

Assessing the reliability of mass-market software (MMS) is difficult even with good reliability feedback data. One challenge is misleading reliability assessments resulting from diverse usage scenarios.

5.1. Windows pre-release reliability assessments

First, we describe some of the metrics and processes used by Windows to assess reliability prior to release. Windows used quantitative measurements and systematic Q/A processes. The Windows Reliability Team measured rate of failures: failures experienced per machine during the first 15 days of runtime. The team evaluated the metrics across time and major releases; several million users tested Windows Vista and Vista SP1 in the beta-to-release timeframe. Windows

compared reliability metrics against a Windows XP® SP2 baseline and triaged and addressed the top failures. Garzia et al. describe this process in detail in [11].

5.2. Reliability measurements

In Figure 4, we show failure rates for 9939 Microsoft internal machines and a comparable sample of 9939 external machines for a pre-release version of Windows Vista and for 2319 Microsoft internal machines and a comparable sample of 2319 external machines for a pre-release version of Vista SP1. The code within the internal and external samples is the same; thus we reduce the possibility of code differences confounding the results. Failure rates are higher in the external sample. This difference is significant at the 95% confidence-level based on a boot-strap confidence interval (an evaluation of measurement variability based on random sub-sampling of the data [10]).

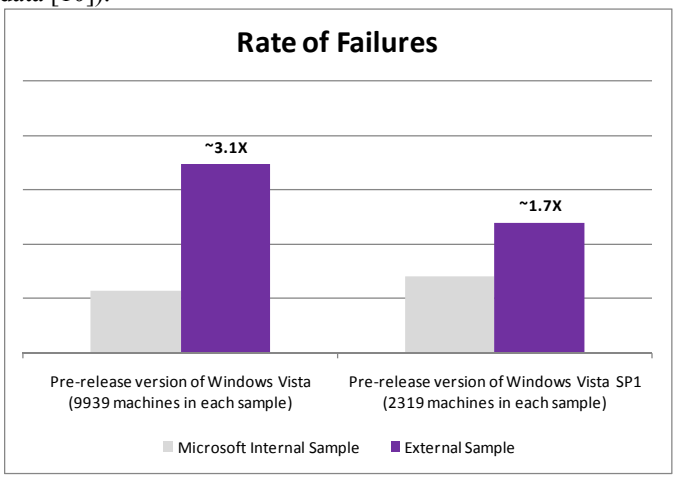

Figure 4. Rates of failures for pre-release versions of Windows Vista and Vista SP1

5.3. Scenario differences

To determine if failure rate differences correspond to usage scenario differences, we characterize usage scenarios using application execution, i.e. whether an application is executed. This is a simple characterization and does not consider execution combinations, execution frequency, or execution purpose (i.e. what users do with the application). Nonetheless, even with this simple characterization we can identify differences: if an application is never executed then scenarios involving the application are not covered and failures in those scenarios cannot be detected.

Data in Figures 5 and 6 indicate significant difference in both the applications executed and the applications that had failures.

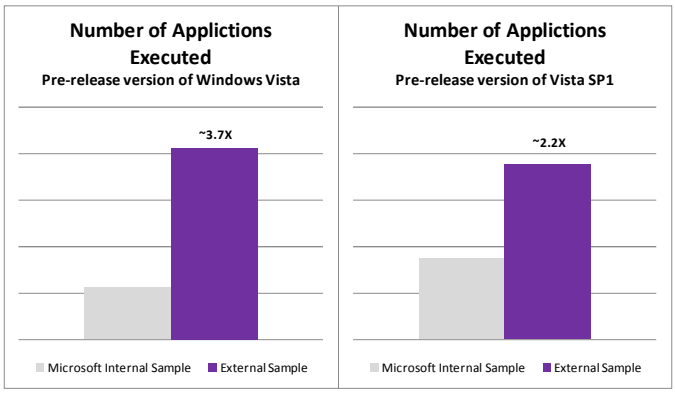

Figure 5. Applications executed in pre-release versions of Windows Vista and Vista SP1

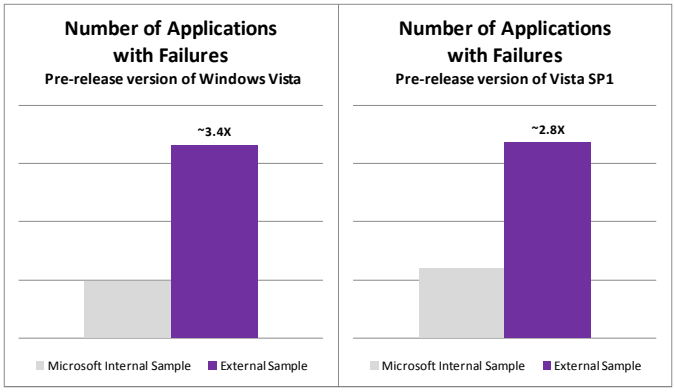

Figure 6. Applications that failed in pre-release versions of Windows Vista and Vista SP1

An alterative explanation for the differences in failure rates is applications failing more often externally. However, data indicate that this may not have contributed significantly. In Figure 7 and 8 we show the failure exit ratio (rate of failures per application launch) for three popular applications for pre-release versions of Windows Vista and Vista SP1. The failure rate of some applications improved, some stayed the same, while others degraded.

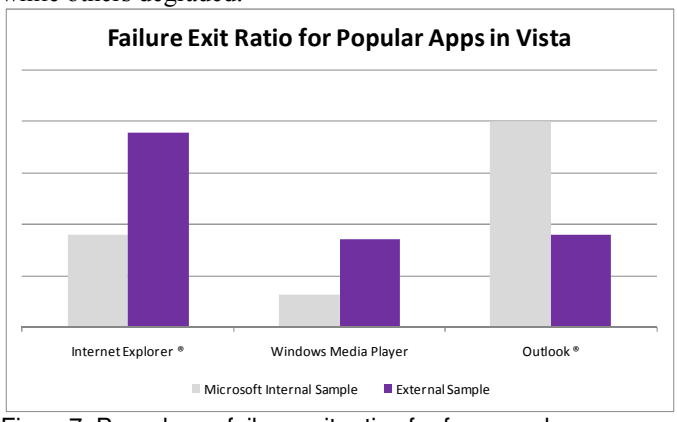

Figure7. Pre-release failure exit ratios for four popular applications in Vista

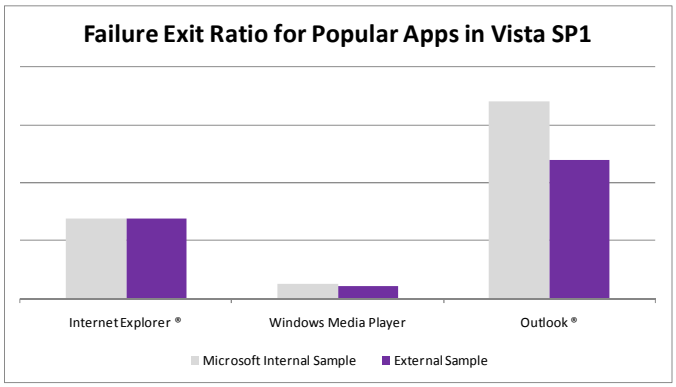

Figure 8. Pre-release failure exit ratios for four popular applications in Vista SP1

There is no evidence that applications consistently fail more externally.

6. Discussion

In this paper, we have discussed two insights gained from assessing the reliability of Windows Vista. First, to meet the quality expectations of customers, software producers should use automated reliability monitoring; it provides normalized reliability feedback data that other approaches cannot provide. Second, producers of MMS need methods for adjusting reliability assessments based on differences in usage scenarios because usage scenarios differences can significantly affect reliability assessments.

The traditional approach of using operational profiles described by Musa [19] may not be feasible because of the complexity of the usage scenarios. The Windows Reliability team is working on ways to further address usage scenario differences.

Our second insight is externally valid even though Windows is likely unique in the number of different usage scenarios. Other operating systems, extendable browsers, and applications that require inter-operability may face similar problems. For example, Li et al. has shown that diverse usage scenarios are problematic for pre-release reliability assessments of a federated database system in [15].

Microsoft is striving to provide all users of mass-market software a better reliability experience.

7. Acknowledgment

We would like to thank the Windows Reliability Team, especially Solom Heddaya, Vince Orgovan, and Greg Nichols, as well as Brendan Murphy, Nachi Nagappan, and Tom Ball of Microsoft Research.

8. References

[1] ABB Inc. *The red line holding technologies together*, http://www.abb.com/cawp/seitp255/359bcb94e511843bc1256c76005a8813.aspx

[2] Andrzejewski, D., Mulhern A., Liblit B., Zhu X. Statistical Debugging Using Latent Topic Models. *Proc. European Conference on Machine Learning*, Sep 2007.

[3] Augustine, V., Podgurski, A. Corroborating User Assessments of Software Behavior to Facilitate Operational Testing. *Proc ISSRE,* 2007, pp 61-70

[4] Buckley, M., Chillarege, R. Discovering relationships between service and customer satisfaction, *Proc. International Conference on Software Maintenance,* 1995, pp 192 – 201

[5] Charles. *Making Windows Vista Reliable: Introduction to Windows Reliability with Mario Garzia* http://channel9vip.orcsweb.com/Showpost.aspx?postid=286934

[6] Chulani, S., Santhanam, P., Moore, D., Davidson G. Deriving a software quality view from customer satisfaction and service data. *Proc European Conference on Metrics and Measurement*, 2001.

[7] Clause, J., Orso, A. A technique for enabling and supporting debugging of field failures. *Proc ICSE*, 2007, pp 261-270

[8] Diep, M., Cohen, M., Elbaum, S. Probe distribution techniques to profile events in deployed software. *Proc ISSRE*, 2006, pp 331-342

[9] Edwards, S. Gendron, M. Lohrenz, M. Electronic Moving Map, *Proc. Industrial Engineering & Management Symposium,* Jul 2003

[10] Efron, B., Tibshirani, R.J. *An Introduction to the Bootstrap.* CRC Press, 1998

[11] Garzia, M., Khambatti, M., Ni, M. Assessing End-User Reliability Prior To Product Ship. *Proc Reliability Analysis of System Failure Data,* Mar 2007, http://www.deeds.informatik.tu-darmstadt.de/RAF07/program.html

[12] Goseva-Popstojanova, K., Singh A., Mazimdar S., Li, F. Empirical Characterization of Session-based Workload and Reliability for Web Servers, Empirical Software Engineering Journal, Vol 11, No 1, Jan 2006, pp 71-117

[13] Jeske, D.R., Akber-Quereshi, M. Estimating the failure rate of evolving software systems. *Proc ISSRE*, 2000, pp 52-61

[14] Li, P.L., Shaw, M., Herbsleb, J., Ray, B., Santhanam, P. Empirical evaluation of defect projection models for widely-deployed production software systems. *Proc FSE*, 2004, pp 263-272.

[15] Li, P.L., Nakagawa, R., Montroy, R. Estimating the Quality of Widely Used Software Products Using Software Reliability Growth Modeling: Case Study of an IBM Federated Database Project. *Proc. ESEM*, Sept 2007, pp 452-454

[16] Lyu, M. *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

[17] Mockus, A., Zhang, P., Li, P.L. Predictors of customer perceived software quality. *Proc ICSE*, May 2005. pp225-233

[18] Murphy, B. Automating software failure reporting. *ACM Queue*. Vol 2, No 8, Nov 2004, pp 42-48.

[19] Musa, J.D. Operational profiles in software-reliability engineering, *IEEE Software,* Vol 10, No 2, Mar 1993, pp 14-32

[20] National Institute of Standards and Technology. The economic impacts of inadequate infrastructure for software testing. *Planning Report 02-3*, Jun 2002

[21] Ostrand, T., Weyuker, E., Bell, R. Where the bugs are. *Proc ISSTA*, Jul 2004

[22] Schneidewind, N.F., Keller, T.W. Applying reliability models to the space shuttle, *IEEE Software*,Vol 9,No 4, Jul, 1992, pp 28-33

[23] Sharma, VS., Jalote, P. Stabilization Time - A Quality Metric for Software Products. *Proc ISSRE*, 2006, pp 45-51

[24] Weyns, K., Martin, H. Sensitivity of website reliability to usage profile change. *Proc. ISSRE,* 2007, pp 3-8

[25] Windows. *Customer Experience Improvement Program* http://www.microsoft.com/windowsvista/privacy/vistartm.mspx

[26] Windows. *Introducing Windows Error Reporting*. http://msdn2.microsoft.com/en-us/isv/bb190483.asp