Investing in Software Testing: The Risks to System Quality

Abstract

Before we can build a high-fidelity test system, we have to understand what quality means to our customers. Test professionals can avail themselves of three powerful techniques for analyzing risks to system quality. Targeting our testing investment by increasing effort for those areas most at risk results in the highest return on investment.

Introduction

In the previous article, I stressed the importance of a high-fidelity test system; i.e., one that allows the test team to forecast the customer's experience of quality after release. How can you create such test systems? Start by understanding what quality means to your customers.

Quality and Quality Risks

Quality, while considered an amorphous concept by some, is actually a well-defined notion in the field of quality management. In *Quality Is Free*, Phil Crosby describes quality as "conformance to requirements." But what if your requirements-gathering process is poor or non-existent? Many of us have worked on projects with such questionable requirements. In *Software Requirements*, Karl Wiegers identifies a variety of common problems in software requirements that afflict many projects.

Since I can't always count on requirements, I prefer J.M. Juran's definition of quality. He spends a goodly portion of an early chapter in *Juran on Planning for Quality* discussing the meaning of quality, but he also offers a pithy definition: *fitness for use*. In other words, quality exists in a product—a coffee maker, a car, or a software system—when that product is fit for the uses for which the customers buy it and to which the users set it. A product will be fit for use when it exhibits the predominant presence of customer-satisfying behaviors and a relative absence of customer-dissatisfying behaviors.

Armed with this definition of quality, let's move to the topic of risk. Myriad risks—i.e., factors possibly leading to loss or injury—menace software development. When these risks become realities, some projects fail. Wise project managers plan for and manage risks. In any software development project, we can group risks into four categories.

Financial risks. How might the project overrun the budget?

Schedule risks. How might the project exceed the allotted time?

Feature risks. How might we build the wrong product?

Quality risks. How might the product lack customer-satisfying behaviors or possess customer-dissatisfying behaviors?

Testing allows us to assess the system against the various risks to system quality, which allows the project team to manage and balance quality risks against the other three areas.

Classes of Quality Risks

It's important for test professionals to remember that many kinds of quality risks exist. The most obvious is functionality: Does the software provide all the intended capabilities? For example, a word processing program that does not support adding new text in an existing document is worthless.

While functionality is important, remember my self-deprecating anecdote in the last article. In that example, my test team and I focused entirely on functionality to the exclusion of important items like installation. In general, it's easy to over-emphasize a single quality risk and misalign the testing effort with customer usage. Consider the following examples of other classes of quality risks.

Use cases: working features fail when used in realistic sequences.

Robustness: common errors are handled improperly.

Performance: the system functions properly, but too slowly.

Localization: problems with supported languages, time zones, currencies, etc.

Data quality: a database becomes corrupted or accepts improper data.

Usability: the software's interface is cumbersome or inexplicable.

Volume/capacity: at peak or sustained loads, the system fails.

Reliability: too often—especially at peak loads—the system crashes, hangs, kills sessions, and so forth.

In *Managing the Testing Process,* I provide an extensive list of quality risks. Peter Neuman, in *Computer-Related Risks*, catalogs a broad range of system failures that have cost time, money—and even lives. Cem Kaner, Jack Falk, and Hung Nguyen include an appendix of 400 common software errors in *Testing Computer Software*. Boris Beizer offers a taxonomy of bugs in *Software Testing Techniques*. The clever tester can construct a vast catalog of nightmare scenarios for any but the most trivial software, but you probably don't have the time—or the money—to test them all, nor should you.

Tailoring Testing to Quality Risk Priority

To provide maximum return on the testing investment, we have to adjust the amount of time, resources, and attention we pay to each risk based on its priority. The priority of a risk to system quality arises from the extent to which that risk can and might affect the customers' and users' experiences of quality. In other words, the more likely a problem or the more serious the impact of a problem, the more testing that problem area deserves.

You can prioritize in a number of ways. One approach I like is to use a descending scale from one (most risky) to five (least risky) along three dimensions.

Severity. How dangerous is a failure of the system in this area?

Priority. How much does a failure of the system in this area compromise the value of the product to customers and users?

Likelihood. What are the odds that a user will encounter a failure in this area, either due to usage profiles or the technical risk of the problem?

Many such scales exist and can be used to quantify levels of quality risk.

Analyzing Quality Risks

I'm familiar with three techniques for quality risks analysis. The first technique is informal, and works well regardless of development process maturity. To use it, list the quality risks that might apply to your software in the leftmost column of a table. Add a middle column for associated failure modes, and describe the kinds of problems that can occur within that class of quality risk. Keep the descriptions—and the table—short. Now, add a blank rightmost column for priority.

Properly done, the table becomes the map for an exploratory dialogue, either one-on-one, as a group meeting, or via e-mail, with the key stakeholders in the project. I seek input from people like development managers, product architects, business analysts, technical or customer support managers, project managers, sales and marketing staff, operations personnel, and, if available, customers and users. In this dialogue, I solicit a priority number—possibly using a three-part scale as discussed above—for the risks and failure modes, along with delving into any quality risks that I overlooked. Table 1 below shows a fragment of a quality risk analysis for a hypothetical word processor.

Quality Risk	Failure Mode(s)	Priority
Functionality	Can't edit text.	1
	Can't format text.	1
	Can't handle tables.	2
	Can't insert pictures.	3
Performance	Display more than two keystrokes behind.	1
	File ops longer than two seconds for large typical file.	1
	File ops longer than five seconds for large atypical file.	3
Compatibility	Can't import Word files.	1
	Can't import WordPerfect files.	3

Table 1: An Informal Quality Risk Analysis for a Hypothetical Word Processor

A slightly more formal approach is the one described in the International Standards Organization document ISO 9126. This standard proposes that quality of software system can be measured along six major characteristics:

Functionality. Does the system provide the required capabilities?

Reliability. Does the system work as needed when needed?

Usability. Is the system intuitive, comprehensible, and handy to the users?

Efficiency. Is the system sparing in its use of resources?

Maintainability. Can operators, programmers, and customers upgrade the system as needed?

Performance. Does the system fulfill the users' requests speedily?

Within these six characteristics, the ISO 9126 process asks that stakeholders identify the key subcharacteristics for their system. For example, what would it mean for your system to have quality in the area of performance? For a web application, it might mean giving an initial responses to user input with a half-second, completely displaying each page within 30 seconds, and allowing the typical user to complete an e-commerce sales within two minutes. As with the informal quality risks analysis process, once the quality subcharacteristics are identified, stakeholders should determine the priority levels associated with testing each area.

On very structured development projects, you can formalize the quality risks analysis process by using Failure Mode and Effect Analysis (FMEA). Figure 1 shows an example of a FMEA chart for a hypothetical word processor. Ideally, all the stakeholders perform the FMEA in a group meeting. Once again, you should prioritize the myriad quality risks to identify the most important areas for testing. In a FMEA chart, the Risk Priority Number is the product of three factors, Severity, Priority, and Likelihood, as describe above.

Failure Mode and Effects Analysis (Quality Risks Analysis) Form

System Name: SpeedyWriter (with Document Management System add-on) | Model/Product: SpeedyWriter Release 3.1 (w/ DMS)
Product Manager: Kate Hernandez | Target Release Date: 102003
Project Manager: Jenny Kaufmann | Prepared By: Jamal Brown
Other Stakeholders: Bobbi McDaniels (Dev), Max del Oro (Mkt), Casimo Nogroev (Sport | FMEA Date: September 11, 2002
Test Manager: Jamal Brown | FMEA Rev Date: N/A

Risk ID Number	Quality Risk Category	Failure Mode/Quality Risk/Effect	Severity	Priority	Likelihood	Risk Priority Number	Recommended Action	Who/ Which Phase (Unit, Component, Integration, System)?
1.000	Functionality	Failures that cause specific features not to work						
1.001		Regression of existing SpeedyWriter features.	1	1	3	3	Rerun entire R3.0 test set	Test/S
1.002		Can't cancel incomplete actions using cancel or back.	2	3	4	24	Opportunity testing	N/A
1.003		Can't get and select from list of managed files	2	1	3	6	Extensive testing	Dev/UC-Test/IS
1.004		Can't get and select from a list of non-managed files	2	1	4	8	Balanced testing	Dev/UC-Test/IS
1.005		Check-in of new document to DMS fails	2	1	2	4	Extensive testing	Dev/UC-Test/IS
1.006		Check-in of existing document to DMS fails	1	1	2	2	Extensive testing	Dev/UC-Test/IS
1.007		Check-out of DMS documents fails	1	1	2	2	Extensive testing	Dev/UC-Test/IS
1.008		Load of on-line DMS document fails.	1	1	3	3	Extensive testing	Dev/UC-Test/IS
1.009		Load of near on-line DMS document fails.	1	2	3	6	Extensive testing	Dev/UC-Test/IS
1.010		Load of off-line DMS document fails.	1	2	3	6	Extensive testing	Dev/UC-Test/IS
1.011		Can't display or modify doc properties in DMS before saving.	2	3	5	30	Opportunity testing	N/A
Quality Risk Category Priority						2		
2.000	Load, Capacity, and Volume	Failures in scaling of system to expected peak concurrent usage levels						
2.001		System fails at or before 25 concurrent users	1	1	3	3	Extensive testing	Test/S
2.002		System fails at or before 255 concurrent users	1	3	2	6	Extensive testing	Test/S
2.003		System disallows 255 or fewer user accounts.	1	1	3	3	Extensive testing	Test/S
2.004		System disallows 32,767 or fewer user accounts	1	3	3	9	Balanced testing	Test/S
2.005		System fails on documents larger than 100 KB	1	1	1	1	Extensive testing	Test/S

Figure 1: Failure Mode and Effects Analysis for a Hypothetical File Server

Any of these techniques can become time consuming, confusing, and cumbersome if you try to go into too much detail in the failure modes or quality subcharacteristics. Focus on relatively high-level descriptions. "Misspelled user messages," is good. Trying to list every possible misspelling that could occur in each error message would be bad. The objective is not to try to document bugs you haven't even found yet, but rather to focus test development and execution activities.

Not every quality risk can be a high priority. When discussing risks to system quality, I don't ask people, "Do you want us to make sure this area works?" In the absence of trade-offs, everyone wants better quality. Setting the standard for quality higher requires more money spent on testing, pushes out the release date, and can distract from more important priorities—like focusing the team on the next release. To determine the real priority of a potential problem, ask people, "How much money, time, and attention would you be willing to give to problems in this area? Would you pay for an extra tester to look for bugs in this area, and would you delay shipping the product if that tester succeed in finding bugs?" While achieving better quality generates a positive return on investment in the long run, as with the stock market, you get a better return on investment where the risk is higher. Happily, unlike the stock market, the risk of your test effort failing does not increase when you take on the most important risks to system quality, but rather your chances of test success increase.

Whatever risk analysis process you chose, at the end you should have a ranking of risks (or quality subcharacteristics) by priority. These risk priorities will help you winnow down the test effort from everything you conceivably *could test* to some manageable, agreed-upon list of what you *should test*. You should test the

high priority risks items extensively, the medium priority risk items broadly, and the low priority risks items in a cursory fashion. The items that pose little if any risk to the quality of the system should receive none of your scare time, resources, and attention at all. For more information on these quality risk analysis techniques, see my first book, *Managing the Testing Process*, my new book, *Critical Testing Processes Volume I*, or D.H. Stamatis' book, *Failure Mode and Effect Analysis*.

Building on Your Foundation

A well-defined, properly prioritized list of quality risks is the foundation of your test effort. Understanding the importance of each quality risks to your customers allows you to build a high-fidelity test system, predictive of the customer's experience of quality. In construction, sound building techniques are required to construct a trustworthy house on a good foundation. Likewise, you now need to apply sensible test design and implementation methodologies to the creation of your test system. In the next article, I'll review some of the common techniques test engineers can use to build such test systems.

Author Biography

Rex Black is President and Principal Consultant of RBCS, Inc. (www.RexBlackConsulting.com), a consultancy that provides testing experts world-wide, serving clients such as Bank One, Cisco, Hitachi, IMG, and Schlumberger in consulting, training, and hands-on implementation. He's written *Managing the Testing Process, Critical Testing Processes Volumes I* and *II*, and numerous articles, along with presenting papers and keynote speeches at international conferences.

Bibliography

International Standards Organization, *ISO/IEC 9126:1991 (E)*. Geneva, Switzerland: ISO/IEC Copyright Office, 1991.

B. Beizer, *Software Testing Techniques, 2nd Ed*. New York, Van Nostrand Reinhold, 1990.

R. Black, *Critical Testing Processes, Volume I*. New York, Addison-Wesley, 2002.

R. Black, *Managing the Testing Process, 2nd Ed*. New York, Wiley, 2002.

P. Crosby, *Quality is Free*. New York, Mentor, 1980.

J. Juran, *Juran on Planning for Quality*. New York, The Free Press, 1988.

C. Kaner, et al., *Testing Computer Software, 2nd Ed*. New York, Wiley, 1999.

P. Neumann, *Computer-Related Risks.* New York: Addison-Wesley, 1995.

D. Stamatis, *Failure Mode and Effect Analysis*. Milwaukee, ASQC Quality Press, 1995.

K. Wiegers, *Software Requirements*. Redmond: Microsoft Press, 1999.