Development and project delivery metrics
White paper
September 2008

Rational. software

IBM

When am I done testing?

A new barometer for measuring the risk of releasing new software versus the cost of continued testing

Murray Cantor, distinguished engineer, Rational software, IBM Software Group

Michael Lundblad, program manager, quality management, Rational software, IBM Software Group

Avik Sinha, research scientist, test automation, software measurement and estimation, IBM Research

Clay Williams, manager, governance science research, IBM Research

Contents

Introduction

A rough estimate puts the costs of software-related downtime at about US$300 billion annually.[1] Consider these examples:

- *In September 2007, a computer system in Atlanta that processes pilots' flight plans and sends them to air traffic controllers fails. The agency reroutes the system's functions to another computer in Salt Lake City, which overloads because of the increased volume of data, magnifying the problem.*[2]
- *A telecommunications provider spends US$3 million on software application support over six months instead of on new application development.*[3]
- *A health insurance company loses US$20 billion annually in lost business and repair work related to production performance problems.*[4]

Research and experience have shown that the cost of finding and removing defects grows exponentially with the quality of the code. That is, as each defect is removed, the cost of finding and removing the next defect increases. Hence, it is a practical impossibility to deliver zero-defect software. C-level executives understandably want products whose defects have a minimal impact on users—either functionally or financially. But such certainty—and the enhanced ability to avoid disasters like those described above—costs money. Hundreds of millions of dollars are spent yearly on software testing. Indeed, IBM's direct experience with organizations worldwide and data gathered by researchers working with hundreds of companies show that most firms invest 25 percent or more of their development lifecycle time and cost in quality assurance.[5]

Highlights

A key point in any development process, therefore, is the point at which testing ends and the organization moves ahead with deployment. This is the time in the life of every development project when the program manager has to ask the very practical questions:

- *"Is it wise to continue spending money on quality assurance?"*
- *"Will further testing cost more than it is worth?"*
- *"Is the software ready for release?"*
- *"How do we know when we're done testing?"*

Project managers need a way to measure the business risk of release versus the cost of continued testing.

This white paper provides an innovative framework for answering those questions—a new barometer for measuring the business risk of release versus the cost of continued testing.

Framing the right question—technical versus economic risk

Upon initial examination, the question of when to stop testing appears to be solely technical. Organizations commonly apply exit criteria for the testing process based on factors such as the percentage of successful tests for completeness of functionality and the number of defects remaining at various severity levels. Some quality assurance teams have also measured quality using metrics in areas such as:

- *Defect density and glide path.*
- *Requirements volatility.*
- *Code churn.*
- *Mean time to failure (*MTTF*) and repair in regression and load/stress testing.*
- *Test coverage.*
- *Usability, reliability, performance/scalability and supportability.*

Highlights

Technical measures alone can miss key economic implications of testing versus release.

These technical testing measures are critical, but they are often blindly applied. Applied without a larger context, they can miss key economic implications inherent in the decision to continue testing or to release the software.

Instead of applying only technical criteria, the decision to release must also consider the timing of the window of opportunity that surrounds the software's release date. Late release can mean lost revenue or lost efficiencies. However, early release can mean risk to the business in damaged reputation, organizational disruption and high service costs. To find this balance, companies should consider a number of factors, including:

- *The amount of improvement that can be expected with more investment in testing.*
- *Economic risk reduction versus the lost economic benefit of release over time.*

A basic formula begins by calculating expected cost avoidance and expected value lost.

Getting started with a basic formula

Suppose that for a given software build it is possible to approximate the company's loss in revenue and maintenance to fix a software failure. Call that the *expected cost avoidance (ECA)*. With more testing, one would expect the *ECA* to decrease. If not, why spend money on testing?

However, with more testing, the release date moves out, resulting in lost benefit to the business. So we also need to measure this monetary lost benefit. Call that the *expected value lost (EVL)*. *EVL* can increase with new functionality but decrease with time to release.

Highlights

Any investment in further testing must consider both technical and economic issues.

Early in the development cycle, the *ECA* presumably is much higher than the *EVL*. The product does not do much, but it would generate lots of defects to fix. As a result, the *expected business risk (EBR)*, calculated as $EBR = ECA - EVL$, is a large positive number. Eventually, with good execution, the *EBR* should go to zero or even turn negative.

Even so, the question remains: "Does the reduction in *EBR* justify further testing?" Call the cost of testing, C_T. As the ratio of EBR/C_T approaches 1 or less, it becomes apparent that the investment in further testing is not yielding any more value and might even be counterproductive.

Analyzing usage for a more precise answer

The above reasoning presupposes that we can actually determine *EBR*. As we will show, there are practical, if not simple, methods to compute business risk in terms of *ECA* and *EVL*. To apply the reasoning, it is necessary to thoroughly analyze the system usage to establish the right test cases that not only provide sufficient usage and code coverage, but also test those other considerations such as security, performance and reliability that have economic impact after deployment. Test planning, as a result, should include both technical and economic concerns.

Highlights

Determining economic risks from quality concerns

When considering the economic risks and value of deployment, a useful starting point is the common set of quality concerns known as *FURPS—features, usability, reliability, performance and supportability*:

- ***Features***—*Did we provide the most value? Most testing organizations do a fairly decent job of functional and regression testing.*
- ***Usability***—*Can users accomplish the defined work efficiently? Normally, this area is examined through end-user acceptance testing.*
- ***Reliability***—*Are the frequency of and recovery from software crashes in test or production adding unreasonable operational or liability costs?*
- ***Performance***—*Are the software response times, throughput or both sufficient to meet business productivity needs?*
- ***Supportability***—*Can the defects be isolated and removed economically?*

Each* FURPS *concern has technical and economic implications; to establish patterns, this paper examines reliability.

Each of these concerns has both technical specification and economic implications. It is beyond the scope of the paper to fully address them all. Rather, we can apply the reasoning to *reliability* to establish the pattern that, with adjustment, can be equally applied to other concerns. Therefore, we start with the questions: "What does reliability look like to the business?" and "What is the cost of a system failure?"

Highlights

The failure density function can be used to extrapolate the failure rate beyond practical testing time.

The equation applied to reliability testing

The equation for reliability testing has two aspects—determining of *ECA* and determining *EVL*. In the sections that follow, we evaluate both issues.

Using the failure density function (*fdf*) to determine *ECA*

To measure the statistical impact to business risk in terms of *ECA*, some theory is needed to extrapolate the failure rate beyond practical testing time. This theory is available using the *failure density function* (*fdf*), which is described by a statistical distribution. Unlike the bell curve that describes normal distribution, the *fdf* distribution for likelihood of failures over time[6] is:

$$= \lambda e^{-\lambda t} \quad \text{for } t \geq 0$$
$$= 0 \qquad \text{for } t < 0$$

The graph of *fdf*'s for different build λ's is given by figure 1 below, where the *x* axis shows time and the *y* axis shows the density of failures at a given time. Note that the probability of a failure before a given time (*t*) is given by the area under the curve from 0 to *t*. Applying some first-year calculus results in the following:

$$P[0,t] = 1 - e^{-\lambda t}$$

where $P[0,t]$ is the likelihood of failure before time *t*.

Highlights

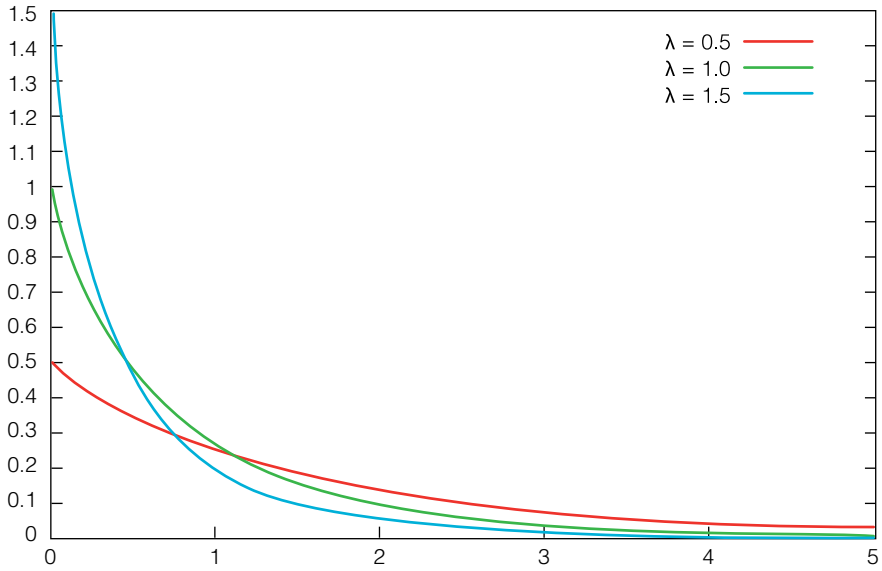

Figure 1. Some failure density functions (IBM illustrative example)

The program is more reliable when failures are more likely to arise further in the future.

Testing and repair should decrease the λ—in effect flattening and pushing to the right the failure density function distribution. Decreasing λ then means the failures are more likely to arise further in the future, making the program more reliable.

To get an approximate λ for a given build:

- *Run multiple system tests under a variety of conditions and loads. The required number of tests can be time consuming and costly, so the team must develop means for efficiently creating enough runs with different loads and orders of functional invocations to create the distribution. There are two common approaches: automated testing and volume beta testing. In both cases, it is necessary to capture the time to failure for each run and the debugging data at the failure.*

Highlights

- *Create a histogram of the times to failure. Normalize the histogram by dividing each of its values by the number of tests. This results in a table showing percentage of failure rates over time.*
- *Curve-fit the table to get an* fdf *distribution.*

Once λ is known, it is possible to estimate the likelihood of failure between time 0 and any time *t* by computing the area under the curve from 0 to *t*. Note this is true even if the time is far in the future.

From each of the build *fdf* distributions, we can apply the business loss probability and maintenance impact measures to get the *ECA*.

The value of a program at release generally is time dependent—the later the release, the less the program is worth.

Finding the value of software at release—determining *EVL*

To measure the *EVL*, the organization must have some idea of the value of the program at release. This value generally is time dependent—that is, the later the release, the less the program is worth. Some examples include:

- *Contracted delivery with penalty clauses, awards fees based on acceptance or both.*
- *Delivery of a system to the marketplace when the profit depends on the timing of introduction.*

In the latter case, data must be provided by the marketing department, as there are other ways in addition to timing that the system might deliver value.[7] In any case, a monetized value at delivery is necessary to making the "go/no-go" decision regarding testing.

Highlights

It is necessary to ask, at the beginning, how a business or industry defines product reliability.

Instrumenting the equation for reliability

The monetized business losses that result from reliability failures are directly related to the sort of software or system under development and the industry for which it is being developed. The starting point, therefore, is to ask the question, "What does reliability look like to the business?"

A simple example is one that many individuals and businesses encounter daily: a laptop operating system (OS) is often restarted after several days of using many software programs simultaneously to avoid abnormal behaviors such as slow performance, screen errors or even locked screens that may occur later as a result of continuous use. The developer of the OS knows that the cost of failure is low enough and that the OS meets standards of acceptable reliability if the *MTTF* is greater than the acceptable time between restarts.

Another example occurs less frequently and can be more costly: If the system is an office telephone switch, each failure may cost tens of thousands of dollars. In this case, the likelihood of a single failure in a year needs to be less than 10^{-3} or 10^{-4}.

Using the failure density function analysis described earlier, we now examine an example from the IBM Research Lab to substantiate the premise.

Measuring reliability as time to failure

Reliability is measured as the time between failures during testing and field operations. To test reliability, a large number of simultaneous test runs for a series of software builds is necessary under a variety of loads and random functions. The time to failure can vary with each build. Thus for each build we can create a histogram of time to failure. Each build has a different histogram.

Highlights

For example, a given build may use automated testing to run 10,000 reliability tests. Each test case launches the build into a parallel process. The number of processes that crashed during a day's run are counted and recorded. The whole experiment is observed for a period of four days as illustrated in this table:

	Build 1	Build 2	Build 3	Build 4	Build 5
Day 1	31	10	5	3	2
Day 2	125	49	24	13	10
Day 3	453	233	112	63	42
Day 4	5,896	1,333	650	330	210

Table 1. Frequency of failures for each build (IBM illustrative example)

It is possible to extrapolate and compute the probability of failure for any interval of future time.

Although reliability experiments ran for only four days (and some runs did not fail in the time allotted), it is possible to extrapolate and compute the probability of failure for the system for any interval of future time.

Applying the extrapolation process to the data in table 1 results in the following graph:

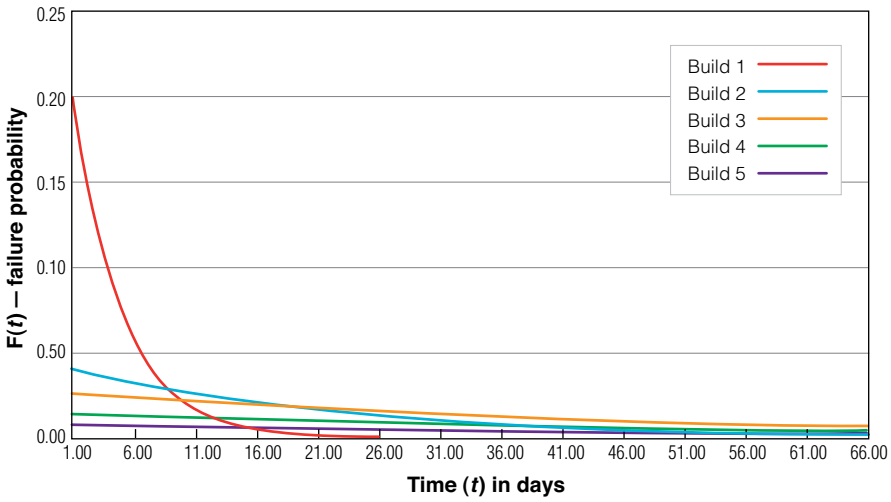

Figure 2. Extrapolated probability of failure (IBM illustrative example)

Highlights

For build 1, $\lambda = 2.63 \times 10^{-3}$, and the chance of system failure for a continuous period of up to one month is determined by computing the area under the extrapolated curve according to the formula $F(t) = 1 - e^{-\lambda t} = 99.96\%$. Note that it is possible to estimate this chance of failure even though the test may not last for the full one-month period.

Note also that additional measures of the reliability improvements from testing and repair activity can be measured by various statistical measures, e.g., the mean of the build *fdf* distributions. For example, the *MTTF* is the mean of the *fdf* distribution for a given build. A computation of *MTTF* for builds 1–5 is shown in table 2.

	Build 1	Build 2	Build 3	Build 4	Build 5
MTTF (days)	3.80	22.56	48.54	95.79	149.51

Table 2. Extrapolated mean time to failure (IBM illustrative example)

Once the likelihood of failure is known, it is possible to estimate the costs of addressing that failure.

Measuring business risk

Now let's consider the economic risks of testing. Once we know the likelihood of failure in any time period, we can also estimate the costs of addressing those failures. These costs depend on a variety of business variables such as the cost of labor and the purpose of the system. For each build, we need to estimate the following:

- *Expected maintenance expenses if shipped now* (m)*—calculated from cost of repair multiplied by frequency*
- *Expected business loss due to system failure if shipped now* (bl)

Highlights

The business losses bl_0, bl_1 and maintenance expenses m_0, m_1 at times t_0 and t_1 are random variables, because they can only be estimated. With an assumed business loss basis of US$20 million, multiplying US$20 million by the probability of failure at the interval of time provides the *bl* variable. The maintenance stream is also provided as an estimated assumed set of variables.

To continue the example, by adding the expected (i.e., mean) maintenance and business loss expenses, we can expand table 2 to table 3.

	Build 1	Build 2	Build 3	Build 4	Build 5
MTTF (days)	3.80	22.56	48.54	95.79	149.51
Maintenance cost	US$1,300,000	US$650,000	US$1,000,000	US$300,000	US$200,000
Business loss	US$19,992,46	US$14,710,482	US$9,219,881	US$5,377,919	US$3,636,197

Table 3. Business risk of deployment (IBM illustrative example)

From these random variables, it is possible to determine the expected losses due to failures in reliability. For increased testing, the *ECA* is determined by:

$$ECA = \text{mean}(bl_1 + m_1) - \text{mean}(bl_0 + m_0)$$

When the economic benefits and risk reduction of testing are leveling off but not flat, it is necessary to consider how much benefit is lost by postponing release.

This data suggests that we may have finished testing, but we cannot be sure. The economic benefits and risk reduction of testing are leveling off, but they are not flat. We also need to consider how much benefit in terms of new revenue is lost by postponing release. To do that we need to consider the value of the system itself for each build.

Highlights

Because the system or software may deliver value over some future interval of time, the *net present value (NPV)*—which we will refer to as *net present revenue (NPR)* of the system or software at delivery—must be computed. A useful form of the equation for this purpose is:

$$NPV = \sum_{i=1}^{n} \frac{R_i}{(1 + r_R)^i}$$

Where R_i is the expected stream of revenues or benefits (say, quarter by quarter) and r_R is the interest rate associated with the time value of money (say, the treasury bill rate). Generally for development efforts, each of the R_i is a random variable, like the *fdf*.

Considering net present revenue allows balancing the cost avoidance achieved by increasing testing with the loss of value from delays.

The important consideration for the *NPR* of the system is that it tends to decrease with the delay in delivery but might increase with more functionality. Considering *NPR* allows us to balance cost avoidance achieved by increasing testing with the loss of value from delays due to testing.

Returning to our example, if we are told by the marketing department that the R_i for quarterly estimates is US$5 million, and $r_R = 6\%$, *NPR* for the system at different delivery dates is shown in table 4.

	2008—2Q	2008—3Q	2008—4Q	2009—1Q	2009—2Q
∑ (*NPR*)	US$21,061,818	US$17,325,528	US$13,365,060	US$9,166,963	US$4,716,981

Table 4. NPR (IBM illustrative example)

Highlights

Similarly, the *EVL* due to missed opportunities during the testing period is determined by:

$$EVL = \text{mean}(NPR_0) - \text{mean}(NPR_1)$$

The overall *EBR* of increased testing is:

$$EBR = ECA - EVL$$

For builds 2–5 of the example system, the *ECA*, *EVL* and *EBR* computation, therefore, is as follows:

	Build 2	Build 3	Build 4	Build 5
ECA	US$5,931,987	US$5,140,601	US$4,541,962	US$1,841,722
EVL	US$3,736,291	US$3,960,468	US$4,198,096	US$4,449,982
EBR	US$2,195,696	US$1,180,132	US$343,866	-US$2,608,260

Table 5. ECA, EVL and EBR calculations (IBM illustrative example)

When the expected business risk turns negative, it may be time to stop testing.

Note that the *EBR* turns negative at build 5. This certainly suggests it is time to stop testing. In the next section, we explore the question further.

Criteria for stopping tests—business risk versus cost of testing

With this framework in place, the stage is set to answer the question, "When am I done testing?" Depending on the depth of understanding of the risks and values, one might apply a statistical approach to the random variables to get the answer. To keep the answer as simple as possible, we continue to look solely at the means.

Highlights

By calculating the expected value of testing, it is possible to determine when the organization is spending more than it is saving—and also when an increase in testing can increase savings.

Here are two ways to answer the question:

1. At a certain point in the process, the *EBR* may turn negative. At this point the lost opportunity revenue exceeds the costs being avoided, so it is reasonable to go ahead and ship.
2. Given the C_T, it is possible to determine the expected value of testing (EV_T) as a savings-to-cost ratio:

$$EV_T = EBR/C_T$$

A value less than 1 indicates that the organization is spending more than it is saving—whereas a value greater than 1 indicates a savings due to increased testing. Therefore, in table 6 it would appear that testing might stop at build 5, where $EV_T < 0$.

	Build 2	Build 3	Build 4	Build 5
EV_T	5.63	1.97	1.91	-21.74

Table 6. When to stop testing (IBM illustrative example)

With the EV_T calculation, we have completed the economic analysis. We now find that in this example we are losing more money in the cost of testing and missed potential revenue than we are potentially saving by avoiding the risk of failure. So now we need to ask ourselves, "Are there other reasons for creating another build?" This is not a mechanistic decision, but the data does help guide our thinking. Other factors such as the corporate culture determine the next step. Decision makers may want to elevate their confidence levels with one more build.

Highlights

A governance model helps decide the "who, what, when, why and how" of decision making.

Quality governance: organizational and decision concerns

Finally, it also is important to consider the quality governance aspects of the decision to declare testing "done." Governance is about deciding the "who, what, when, why and how" of decision making.[8] Table 7 summarizes the factors to consider for each aspect of the testing governance model.

Question	Meaning	Factors to consider
Who? (roles)	Who decides we are done? Who is consulted? Who is informed of the decision?	• Chief quality officer • Quality assurance statistician
What? (decisions)	What artifacts from the testing process feed the decision-making process?	• Test plans and results • Graphs
When? (timing and scheduling)	By when must the decision be made?	• *fdf* reviews during system test phase
Why? (policies)	What policies and procedures have been established to guide how the decision should be made?	• $EV_T = EB/C_T$, where $EV_T < 1$, indicates more is being spent on testing than the amount expected to be gained in reducing business risk
How? (artifacts)	What test plans, measures and results enable informed decision making for the business risk analysis?	• *fdf* graphs • Expected cost avoidance $ECA = (bl_1 + m_1) - (bl_0 + m_0)$ • Expected value lost $EVL = \text{mean}(NPR_0) - \text{mean}(NPR_1)$ • Expected business risk $EBR = ECA - EVL$ • Expected value ratio $EV_T = EBR/C_T$

Table 7. Factors in the testing governance model

Highlights

Because the quality of software can affect the entire business, the issue of quality governance often involves C-level and director-level executives. It may be time, however, to consider two new roles that do not exist in most organizations—a chief quality officer and a quality assurance statistician.

A quality "czar" can drive overarching quality strategy and decision making.

Today quality assurance teams focus on testing and provide resulting *FURPS* validation for use in decision making. But someone, a quality "czar" with specific responsibilities for full lifecycle quality management of business-critical software, needs to drive the overarching quality strategy and decision-making criteria for the business.

A quality assurance statistician can provide the analysis required for "go/no-go" decisions.

In addition, traditional quality assurance teams do not have anyone on staff with the mathematical background to perform the analysis discussed in this paper. A dedicated quality assurance statistician, however, can provide the detailed analysis required for informed "go/no-go" decisions.

Finally, these calculations would provide a basis for governance decisions that follow software deployment. For example, *MTTF* and probability data from reliability testing for failure can be extremely useful in planning software and system maintenance cycles. Aircraft, for example, have cyclical maintenance periods based on such data. Failure testing may also reveal useful information to include in end-user documentation for troubleshooting. Failures related to scalability performance can become the basis for production monitoring decisions.

Highlights

Determining the business impact of shipping now versus the cost of further testing can provide a firm basis for knowing when testing should end.

Conclusion

Deciding when testing is complete is both an economic and a technical issue. Common *FURPS*-based testing provides significant insight into the needs, the process and the status of technical results. But for a true understanding of new software's readiness, the business impact of shipping now versus further testing and repair must be considered.

We have shown that the techniques that provide the required insights are available. Not surprisingly, they do require some analytical capability and a new consideration in the governance of the release process. However, by adopting these methods and release governance, the company's management team can achieve a firm, economic basis for answering a fundamental development question, "When am I done testing?"

For more information

To learn more about IBM Rational® quality management products and solutions, please contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/rational/offerings/quality

1 The Standish Group, *Comparative Economic Normalization Technology Study,* CHAOS Chronicles v12.3.9, June 30, 2008.

2 "FAA Computer Glitch Causes Flight Delays," *Tescom*, http://www.tescom-intl.com/site/en/tescom.asp?pi=465&doc_id=2923

3, 4 Forrester Research, "Performance-Driven Software Development," Carey Schwaber, February 2006.

5 Walker Royce, Software Project Management: A Unified Framework, (Addison-Wesley Professional, Indianapolis, 1998)

6 M. Modarres, M. Kaminskiy and V. Krivtsov, *Reliability Engineering and Risk Analysis: A Practical Guide* (Marcel Dekker, Inc., New York, 1998).

7 M. Cantor, "The Value of Development," article in preparation.

8 M. Cantor and J. Sanders, "Operational IT Governance," IBM developerWorks®, May 2007 (http://www-128.ibm.com/developerworks/rational/library/may07/cantor_sanders/index.html).

Acknowledgments

The authors thank David Lubanko, Patrick Mancini and Harold Moss of IBM for useful conversation and review in the preparation of this paper.

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
September 2008

RAW14026-USEN-00