SOFTWARE
RISK MANAGEMENT

THE IMPORTANCE OF BUILDING QUALITY AND RELIABILITY INTO THE FULL DEVELOPMENT LIFECYCLE

KAMESH PEMMARAJU
ED LORD
GARY McGRAW PH.D.
INTRODUCTION: ESSENTIAL SOFTWARE MEETS INTERNET TIME

SOFTWARE TECHNOLOGIES continue to evolve in power and complexity. As a result, we are experiencing the rapid expansion of software into all areas of our business and private lives. Today, software is found in cars, traffic lights, household appliances, communications and transport systems, hospitals, airplanes, medical devices, next-generation payment cards, business supply chains, enterprise management systems, etcetera. Software is truly becoming ubiquitous and essential.

Given that our society has become so dependent on technology, today’s software must work. Software characteristics including reliability, availability, security, safety, and quality continue to grow in importance. These characteristics lie at the core of essential software. In the not-too-distant future, every business, no matter how big or small, and most households in the developed world will rely on essential software.

The consequences of essential software failure can be dramatic. At the extreme, the failure of essential software in a safety-critical system can result in loss of life. From a business perspective, the financial consequences of essential software failure can be severe as well:

- The Standish Group estimates that software problems cost $85 billion in lost productivity in 1998.
- Hershey lost $200 million in revenue in Q3 1999 due to an enterprise software glitch.
- eBay lost $4 million in revenue during a 22-hour period when its systems crashed due to a software problem. The lost revenue cascaded into a loss of investor confidence to the tune of $5 billion in eBay market capitalization.
- The SEC has fielded over 20,000 investor complaints related to software problems in online trading.
- FoxMeyer, teetering on the edge of bankruptcy, is suing SAP for $500 million over software that snarled operations.

Even the high-flying dot coms with little tangible equity are concerning themselves with software quality. When a business spends $10-40 million a month to build brand, it must invest in the protection of that brand with safe, secure, and Cigital. Brand awareness and confidence is all too easily eroded, and often software problems are to blame.

- H & R Block recently had a software glitch that allowed on-line customers to view other clients’ tax returns causing a loss of credibility for this service offering and damaging their business reputation.
- CDUniverse’s reputation was compromised when its software was exploited by a hacker who stole 300,000 credit card numbers and published the information online complete with names and addresses.

The solution is to manage software risk just as any other business risk is managed.
When time-to-market pressures mount, there is often little time available to “do it right” from a software development perspective; even though not doing it right may risk the entire business. Internet time seriously compresses the development lifecycle. Fiercely competitive markets and tight budgets contribute to the added difficulty of developing essential software with desired levels of reliability, security or quality. The lack of experienced software professionals does little to help.

Fortunately, most business people understand how to manage risk. Business executives do it every day when they make calculated decisions. Given the right data about software behavior, managers can control software risk.

At Cigital, our approach is to apply an advanced proven methodology, software technologies, and specialized expertise in software risk identification, mitigation and management. The key insight is not applying technology for its own sake, but for the sake of the business.

At a basic technology level, software bugs lie at the root of essential software failures. Bugs and other software flaws translate into an unacceptable level of business risk. The business risk questions thus map directly into technology questions.

• How can we ferret out the most egregious software bugs that are likely to cause the biggest problems?
• How can we avoid treating quality and reliability as add-on features when they are emergent properties of a complete system?
• What should we do to build in quality and reliability from the beginning?
• How do we hedge our bets and manage our risks throughout the software development lifecycle?
• How do we test for quality and the effectiveness of our solutions?
• How do we determine readiness for deployment?

This paper is devoted to these questions. Our answers, though neither silver bullets or simple band-aids, are based on a proven approach to Software Risk Management℠, battle tested for years in the real world by Cigital’s software experts.

A QUALITY METHODOLOGY FROM 50,000 FEET

Our methodology is practical, hands-on, and results-oriented and is backed by years of real-world experience supported by cutting-edge technology research. Importantly, our approach leverages technology but is not driven by it. The basic premise is simple: design quality and testability into software from the beginning and test for quality at key points throughout the development lifecycle. This makes clear business sense, since in simple eco-
nomic terms, finding and removing bugs in a software system before its release is orders of magnitude less expensive and more effective that trying to fix systems after release. Designing good software is just like designing a complicated building: building things right the first time is always less expensive than trying to fix up a fundamentally flawed design (just ask Frank Lloyd Wright!).

Identifying and resolving software risks early in the game saves both time and money. Early test planning ensures that processes and tools—including test automation tools, design analysis tools, and measurement tools—are applied cost-effectively, at the most critical points during development. Test planning provides a solid foundation for Software Risk Management and identifies major testing tasks, including technology requirements, test approach, resource allocation, test scheduling, and test automation strategy, among other things.

**Assessing Software Risk**

One readily apparent software risk correlates directly with business risk. Releasing or otherwise relying on flawed software leads to no end of problems. Customers abandon vendors. Businesses flounder. Lawyers become involved. If a software package is essential to your business, either because it is your product or because it is your operations platform, the risks may be unacceptably high. In this sense, the software reliability question really boils down to determining how much risk your company is willing to take on by counting on a product that includes damaging, even dangerous, software errors.

Put succinctly, the fundamental software question is one of software risk assessment and management. These issues can be framed in terms of potential payoff and required investment. Sound decisions can thus be made using a marriage of business goals and technology realities.

There is no “sure thing” in business. Likewise there is no “sure thing” in software. The key is to understand technical software risks in terms of business impact, then to address the most pressing risks using sound technology. The earlier that situations tending to force bugs into a system during its creation can be identified, the better they can be mitigated. Of course, any such activity is dynamic in the sense that risks evolve and change according to changing market pressures, technology advances, and business strategy. The good news is that the type of risk management approach we espouse comes naturally to business executives.

Both business risks and technology risks must be identified, ranked in order of severity, and addressed in rank order by well-conceived mitigation techniques. Any sort of severity ranking is clearly a context-sensitive and time-dependent perspective that depends directly on the changing business needs and goals of the system at hand. Thus, the risk analysis product should be regularly revisited.

Starting the process early is important. The earlier in the development process that risks are taken into account, the more efficiently mitigation planning and resource allocation can proceed. Test effort, including test planning, technology choice, human resource allocation, and budgeting, should all be driven by business impact.
Fundamentally, risk analysis helps project leadership choose between a large number of possible testing approaches, and realistically determine which ones should be applied.

A basic outline of a test process based on business-relevant risk analysis looks like this:

- **Identify business requirements**: Understand the business needs driving the creation of a software system that directly impact architecture and technology choices.
- **Analyze risks**: Carry out a risk analysis to illustrate major areas of risk and where the test effort should focus.
- **Generate test requirements**: Analyze, identify, examine, and clarify technology-driven information about project factors, product elements and quality criteria. Based on this information, identify test requirements and determine the scope of testing. Decide which of the many possible test cases should be covered, by determining which ones mitigate the most relevant risks and which existing technologies can be used to automate the process.
- **Create test cases**: Use an appropriate test technique to define input/output data, and environmental conditions for actual testing.
- **Develop test procedures**: Express each test case in the form of a sufficiently detailed test procedure that will enable it to be executed.

Keep in mind that the process of risk identification is not a one-time activity. Risks should be regularly reviewed according to changing business and technology drivers.

**Taking Aim At A Moving Target**

One key advantage of software is its built-in plasticity. Most modern extensible systems rely on software for their flexibility. Products with a software component can be easily customized and adjusted according to end user demand. As a result, many of today’s essential business systems include essential software as a competitive advantage. The resulting flexibility is very attractive to customers who often wish to update their systems painlessly in order to stay on top of their markets and keep their competition at bay. Nothing in life comes for free, and software flexibility is no exception. The cost of software flexibility is a marked increase in the complexity of the product that includes the software by at least an order of magnitude. From a quality perspective, the best way to mitigate the complexity risk might be to freeze the functionality of the system to a set of known base operations. Of course, any such activity cuts against the grain of choosing a software solution in the first place, so it needs to be done carefully.

In general, any system that is designed according to well-understood requirements turns out better than a system thrown together arbitrarily. Defining a broad-based set of system behaviors required by key customers is an extremely useful exercise. For one thing, a set of required behaviors that is well specified can be used directly for test planning. The hard part of this exercise is finding a good tradeoff between flexibility/extensibility desires and reliability/quality requirements. This is where experience and expertise are most useful. A pure turnkey solution (with a rigid set of requirements and a fixed design) is usually not flexible...
enough. But too much flexibility makes testing much harder. We espouse an evolutionary approach that adjusts core requirements carefully and only according to real business needs. An evolutionary approach of this type is a means to achieving the ultimate goal of a high-quality product that competes and wins on a world-class scale.

Our evolutionary approach to requirements management recognizes that few products can be completely designed and/or redesigned to a high level of quality in Internet time. Complete redesign simply takes too long. A program must evolve at a reasonable rate, allowing features to be added and bugs fixed even as improvements are made (according to business needs) in basic architecture and design.

The basic idea is to identify a functional system core and use it as a testing/reliability baseline. By restricting the functionality of the system to a set of known operations critical to satisfy customer needs, we can focus in on the important parts of the system and not waste time wandering among non-essential features. Because the functional system core is reasonably sized, we can analyze it from several key standpoints; from a developer/software architect perspective, from a user perspective, and, most importantly for our purposes, from a quality assurance standpoint.

Identifying a core reduces some of the problems associated with a rapidly moving software target. By focusing attention on a base set of functionality that is deployed by all key customers with little modification, we can up the quality of the entire family of products. Quality of the product family will increase, if only because the act of specifying the base functionality improves the knowledge base of the quality assurance and development teams as to desired system behavior. None of this is surprising to Software Engineers, but we have found that very little engineering is going on in most software efforts.

Of course, creating a specification (even if it applies only to the functional core of an essential software system) takes time and effort. This has a direct impact on time-to-market, and from a marketing standpoint may seem to lessen any competitive advantage that software is supposed to add in the first place. But in fact, identifying a core and maintaining its integrity as software evolves saves time in the end. Reuse has been a pie-in-the-sky goal for years, and this is one way of actually reusing a software nucleus even as systems are allowed to evolve. The existence of a core does not disallow customization or evolution. It simply controls the dynamism. Of course, it’s necessary to clearly document and test any customizations to make sure that the customizations do not affect the functioning of the rest of the product (a non-trivial task).

Moving beyond the functional core is the second stage of our approach. Customizing the application for specific markets or customers is a business necessity. Through the use of the functional core as the specification target, our approach provides a handle on system behavior while also allowing the system to be nimble.

The key to successful application of our approach is controlling customization, especially with reference to how it impacts the functional core. The core specification must clearly identify
software mechanisms a user will access as well as unambiguously demarcate system states, so that while customization and other change is allowed, it occurs only within a framework that reduces the chances of change-introduced error. This strategy helps to control a common error-introduction mechanism and thus leads to a higher quality product line.

Software behavior is easier to measure when requirements and expectations (expressed in terms that relate directly to business needs) are clearly defined. High quality software is software that behaves.

**Why An Independent Software Risk Management Team is Necessary**

At Cigital, we approach Software Risk Management based on what we call the competent developer hypothesis. This hypothesis holds that developers are a competent lot, and for the most part they know how to create systems of reasonable quality. The main problem is not a lack of ability or lack of know-how; it is instead the Internet time factor and its accompanying drastic compression of development schedules. Producing high quality software under the intense time pressure brought to bear in today’s competitive software market is extremely hard.

There are, of course, other factors beyond Internet time that also directly impact quality. While good programmers try their best to produce high quality software, they may also lack the knowledge and training necessary to assess and/or improve the reliability, safety and security of the software. Developers like to concentrate on making stuff work, and don’t appreciate having to test or assess it. Securing software, for example, requires an extensive body of knowledge not often available to developers.

For both of these reasons, an independent Software Risk Management team that works in tandem with the developers is necessary. Most importantly, the team will be responsible for understanding the software from a unique software assurance perspective. They answer the question: does the software behave?

The Software Risk Management team plays two key roles in the development of a high quality, reliable software product. They help get things right in the first place by designing in testability and quality—the assurance role—and they make sure the final product is independently evaluated—the assessment role. Both of these roles can and should be applied throughout the development lifecycle.

An independent Software Risk Management team with a single point of leadership over both roles is the best solution. Sometimes the necessary expertise can be found inside an organization, but more often external experts are required. The problem is that today’s QA practitioners know little about software development, architecture, source code measurement, and design for testability. Cigital fields high-expertise Software Risk Management teams through its Consulting Services division.
Centralized leadership of a Software Risk Management team is important, especially in the case of large software projects. Assurance engineers working on different areas of the same product need a common forum so they work toward common goals and communicate effectively. Central leadership provides an opportunity to share and reuse common techniques, tools, and test frameworks. A useful rule of thumb is to maintain a ratio of three test engineers to one domain engineer.

Centralized leadership has another, more important, effect on system quality. A global view of system quality helps keep assurance and assessment efforts aligned to business goals. This is only possible with a centralized reporting system rooted in assurance professionals spread throughout a project's development teams. Software Risk Management team leaders should report directly to executive management (and not to development heads). This ensures independent, unbiased, and expert test status data is forwarded to business decision makers on a regular basis. A focus on the inter-relations between business goals/requirements and assurance data, and a high-bandwidth path for top-down input from business risk mitigation decisions, are both critical to the success of a large assurance effort. These sorts of decisions are best made by executive management and not development management in order to assure objective treatment of IV&V data.

Problems often arise when a Software Risk Management team reports through project or development managers. Generally speaking, development managers tend to view testing as a means to determine whether their product is good enough to be released inside hard and fast time constraints and do not usually consider whether the software in question has the required performance, reliability, security, and safety features. Their job is getting the product out of the door on time and within budget, and they are usually not much interested in what happens after the product is released to the market. Software product performance and its direct relation to business success and failure is clearly an executive management issue.

Assurance engineers who report to project/development managers are naturally forced to live within the development managers' schedule, feature priority, and resource constraints. Most assurance engineers in this situation find carrying out independent unbiased testing difficult at best, because results often run contrary to their direct manager's interest. No one likes to make the boss mad. Assurance engineers should be charged with pursuing a risk management plan based on testing performance, reliability, safety, security and other essential software factors. Unfortunately development managers often find these goals somewhat too lofty or unattainable, and thus a waste of time or money.

Although individual assurance engineers may be competent in their own right, they perform best with clear guidance, support, and encouragement. Development managers rarely take time to motivate assurance engineers, who they regard as a necessary evil that serves merely to highlight problems. This management problem is compounded by the fact that in most software development organizations, testers inhabit the bottom of the totem pole. Testers who are ignored by their direct management are in no position to escalate concerns to upper management. Thus upper management is often left in the dark as to real software quality, and can-
not properly manage software risk appropriately. To be most effective, assurance engineers should base their reporting on objective exit criteria and software measurement technology, and these data should be put in front of business decision makers.

We’ve established the need for an independent Software Risk Management team with centralized leadership that reports to executive management. But where does such a team come from? The good news is that an independent assurance team need not be made up of high-priced external experts. Often it is good enough to build a team from assets inside an organization made up of quality and testing people who are not involved in code development. The bad news is that testing and quality expertise seems to be a rare commodity these days. Determining whether or not to seek help outside of your organization will depend on what the system you are designing is meant to do and what happens if it fails to operate correctly in the marketplace.

**PRODUCT AND PROCESS ISSUES**

Throughout the 1990s, process improvement has been all the rage in software engineering. The simple premise behind process improvement is that processes that are disciplined, repeatable, and mature will result in better software than ad hoc processes. It is hard to disagree with the basic tenets of the process pundits. However, dirty water can pour from clean pipes. Concentrating on process alone can be likened to building an elaborate plumbing infrastructure and neglecting to check water quality throughout the system. In the end, product behavior is what matters. Clients and end users do not purchase and run a process!

We advocate strong use of process improvement techniques in concert with software product assessment. With reference to our pipes analogy, we advocate testing the water in the pipes at various locations in the system. Starting with a clean water design is a good idea too.

Unfortunately, many companies chant the “process improvement” mantra as if it were a silver bullet for all their software development and quality problems. Most process improvement methodology is built on establishing and following a set of software standards that tend to neglect the final product itself and instead concentrate mainly on the development process. As a result, most of the emphasis in these methods amounts to process assessment (ISO, IEC, CMM, etc). Engineering judgment is thus used to assess the process producing software and then assume that the integrity of the software can be correlated with the quality of the process. This is directly the result of the manufacturing view of quality, where the focus is on “doing it right the first time.” The emphasis in all these process standards is on conformance to processes rather than to specifications (which as we have stated should be stated in terms of business goals and objectives). A standard process is certainly necessary but is not sufficient for good software behavior. There is no hard evidence that conformance to process standards guarantees good products. Many companies are finding out the hard way that good processes don’t always result in quality products.

Even in industries where the development of products is subject to strictly enforced processes and contractual requirements, it is common to see processes being short-circuited in the mad
rush to meet the project deadlines or to meet the budget constraints. Changing market condi-
tions and subsequent changes in requirements as well as Internet time pressures do not help
this situation much.

Problems with a myopic focus on process negatively influence the testing process. For exam-
ple, in the name of schedule acceleration, many organizations routinely violate pre-defined test
entry and exit criteria (if they are even defined to begin with). On first blush, “accelerating”
testing by entering a test phase before the software-under-test is ready may sound like a good
idea (why not get a leg up?), but what happens in practice is that the test team finds a raft of
problems with functions that are known to the developers as “not completely implemented”.
So the results are discounted, even though the tests are treated as complete. Parallelizing test-
ing and development can potentially reduce schedule pressures only if proper version control
mechanisms and high levels of tester-developer communications are maintained throughout
the development cycle.

A similar argument applies for exit criteria. Leaving a test phase before the defect rate has lev-
eled off and the other exit criteria are met simply passes an immature system on to the next
phase.

While a manufacturing view tends to examine the process of creating a product, a product-
centric view of quality looks at the product itself. Product assessment advocates stress the fact
that in the end, what runs on a computer is not the process, but the software product.
Product assessment, including dynamic testing, can contribute to a strong quality argument
because of its ability to provide tangible, objective evidence of good software behavior. Note
that the rigor with which the software must be tested increases exponentially with the desired
quality level; unfortunately, the point of diminishing returns is not easily recognizable. Real-
world testing constrained by schedule/budget pressures may probe a limited portion of the
input space or cover only a limited portion of the software product. Quality assessments
based on the results of such testing will again be far removed from reality.

The answer to the process and product debate is to combine the approaches and reap the
benefits of both.

**A Model For Quality Assessment, Testing, Software Risk Management**

The act of assessing software risk and testing to identified risks is clearly the best way
to approach software quality. After all, quality assessment and testing is, at its core, risk man-
agement. The act of designing and developing essential software is a context-sensitive,
domain-specific, time-dependent activity with direct ties to business goals and necessities.
Approaching software quality as a risk management exercise comes as second nature to busi-
ness decision makers. Development managers and software professionals may not share the
same point of view. That is why executives in charge of business strategy and operations
should be involved in the Software Risk Management loop.
We find in practice that the use of a high-level model that captures the essence of quality analysis and testing is very useful. Using such a model, an assurance engineer can apply a standardized set of quality guidelines and criteria to determine if a given architecture has problems. Assurance engineers can then identify appropriate test techniques oriented toward important quality criteria to ferret out potential problems. Note that this is by no means an easy, automatic solution. The successful application of our methodology requires professional assurance engineers with real-world experience. Even applied by experts, model-directed tests may not result in 100% success, such as finding a carefully hidden bug, though sometimes they do. In the end, there is still much art in software quality.

The biggest benefit of an assurance model is that it provides a framework for consistency of analysis. Such a framework provides clues to a test engineer as to how to do a quality analysis in general, and what kinds of risks to consider. These risks and requirements are driven by business goals. No such model can be absolute or complete; it must be constantly updated according to dynamic business pressures.

**SUMMARY**

Essential software systems are becoming more and more common and are beginning to affect both our core businesses and our daily lives deeply. Software failure in essential software systems is unacceptable; serious implications that result from such failure include loss of life and extreme business exposure. Thus the risks that essential software systems bring to bear must be carefully managed. Good software risk management practices, put in place by expert software assurance engineers, can help minimize software risk.

Kamesh Pemmaraju is a Project Manager at Cigital. Contact him at kamesh@cigital.com. Ed Lord is Vice President of Consulting at Cigital with 20 years experience managing and growing consulting organizations. Contact Mr. Lord at elord@cigital.com. Gary McGraw, Ph.D. is Vice President of Corporate Technology at Cigital and co-author of Software Fault Injection: Inoculating Programs Against Errors (Wiley, 1999). His e-mail address is gem@cigital.com.