Making Essential Software Work

Why Software Quality Management Makes Good Business Sense

Gary McGraw, Ph.D.
The Role of “Essential Software”

In today’s computer-oriented society, software no longer simply supports back-office operations and home entertainment. Instead, software has become the lifeblood of our businesses and has become deeply entwined into our daily lives. The invisible hand of embedded software enables e-business, automates manufacturing and supply chains, and provides instant, worldwide access to information. At the same time, software is moving into our cars, televisions, home security systems and even our toasters. This paper explains why all of this software introduces business risk to the companies that develop and deploy it.

For businesses, the equation is simple: software must work. Nearly every large enterprise in the world relies on essential software, either embedded in its products or driving its business systems and operations. As business’ reliance on software grows, so do the business-related consequences of software failure. The term essential software is used to refer to software that must perform with high levels of reliability and security.

According to Dr. William Powers, VP of Research at Ford Motor Company: "Software is where the problem is today. Today, if you change a line of code, you’re looking at the potential for some major problems. Hardware is very predictable, very repeatable. Software is in much more of a transition state."¹ That Ford Motor Company is worried about software behavior may come as a surprise, but is indicative of the important yet hidden role that software plays in business.

Two trends are driving business’ increasing reliance on essential software:

1. the Internet explosion: Businesses and consumers alike recognize the benefits of using the Internet – and the software that supports it – to transact business, access information and communicate.

2. the rapid move to faster, smaller microprocessors: Companies are embedding powerful microprocessors running complex software into an increasing number of products and processes. These software systems are, in turn, networked together to optimize supply chains, streamline manufacturing lines and provide remote diagnostics capability.

While essential software enables e-business, it also poses significant business risks to organizations that depend on it to be reliable and secure. Essential software failure results in severe business consequences:

- Revenue loss in the millions when software fails or key data are stolen or compromised;
- Brand/reputation damage and severe market impact when software does not work as advertised or security vulnerabilities impact consumer trust;
- Liability costs when consumers cannot complete on-line transactions or when software embedded in airplanes, automobiles, pacemakers or nuclear reactors causes injury or death; and
- Productivity loss when software malfunctions or ceases to function altogether.

The threat of software-related losses provides significant incentive for businesses to manage the risks of essential software failure. For this reason, business risk management must include Software Quality Management (SQM). Business risk management models have traditionally ignored software, largely because software’s impact on business operations was considered minor. In the digital age, however, software failure can lead directly to business failure. As a result, a bulletproof approach to SQM is an absolute necessity.

¹ EETimes, September 25, 1996.
Software Problems are Business Problems
The software quality problem is growing at an alarming rate. The current accepted standard of fixing broken software only after it has failed spectacularly in the field is insufficient to control the growing crisis. Software problems are deeply impacting businesses, demonstrating in no uncertain terms that software problems are indeed business problems.

According to a report issued in June 2002 by the National Institute of Standards and Technology (NIST), defective software costs the U.S. economy an estimated $59.5 billion each year. Of that total, software users incurred 64% of the cost and software developers 36%. NIST suggests that improvements in testing could reduce this cost by about a third – or $22.5 billion – but that testing improvements would not eliminate all software errors.²

Similar statistics released in 2002 by the National Research Council show that U.S. companies spent $175 billion in 2001 to repair damage caused by software defects.

Further data regarding the severe business consequences of software failure can be found in real-world examples taken directly from the headlines:

- **Revenue loss.** Nike's Q3 2000 earnings fell 24% from their projected levels after faulty supply-chain software delayed orders and caused excess inventories. Revenues for that quarter were off by more than $100 million. In the days following the company's announcement of the software problems, Nike's stock prices plummeted, dropping $10 per share.³

- **Brand/Reputation damage.** H&R Block faced a massive consumer backlash after customers' passwords and Social Security numbers were displayed in plain text on the company's website. The problem was caused by a switching error in the website's encryption software.⁴

- **Liability costs.** The parent company of bankrupt pharmaceutical distributor FoxMeyer sued SAP for $500 million over enterprise software failure that allegedly crippled operations.⁵

- **Productivity loss.** The Standish Group estimates that defective software code accounted for 45% of computer-system downtime and cost U.S. companies $100 billion in lost productivity in 2000 alone.⁶

Software Quality Management
Software practitioners champion and use many diverse software development processes, ranging from light (e.g., extreme programming, or "XP") to heavy (e.g., Rational Unified Process, or "RUP"). All development processes share a common framework of stages. Whether these common software lifecycle stages are applied in a cyclical manner, as in the spiral model of development, or a linear manner, as in the waterfall model, practitioners talk about five basic stages: Requirements, Design, Coding, Testing, Maintenance.

SQM is about measuring and improving artifacts produced in each of the software lifecycle stages. Once software risk and software quality are being measured, software processes and the software product itself can be iteratively improved.

Cigital's SQM methodology is a three-pronged approach to helping companies create an SQM environment that continually improves productivity and product quality:

---
⁴ USA Today, February 17, 2000.
⁵ InfoWorld Electric, August 27, 1998.
1. **Risk management.** Proactively identifying, analyzing, and fixing software defects that negatively impact business operations. Decisions regarding which risks to manage and how much to spend on fixes are deeply tied to the business context.

2. **Measurement.** Utilizing product quality measurement (e.g., business impact/probability, defects by development stage, amount and source of rework activity) to provide static and dynamic metrics enabling decision support (including release readiness). This allows executives to deliver quality software products to market as quickly and cost-effectively as possible.

3. **Process improvement.** Analyzing quality metrics over several projects to establish software development process improvements that will boost productivity and “speed-to-quality” on future development projects.

**Early is Better: The Software Lifecycle Matters**

The earlier in the software lifecycle that software risks are identified, the better. There are two primary business reasons for this:

1. Most software defects are introduced early in the software lifecycle (see Figure 1); and

2. The earlier in the software lifecycle a defect is uncovered and fixed, the less it costs (see Figure 2).

![Defects Introduced at Each Stage of Software Development](source: TRW)

*Figure 1: Software defects introduced in various lifecycle stages. More defects are introduced early in the software process than later.*
The Trinity of Trouble

Three technical trends are exacerbating the software quality problem, making business impacts both more common and more serious. These three trends can be summed up as follows:

- All modern software is exposed to the Internet;
- Extensible systems (e.g., Java and .NET) are dangerous; and
- System complexity is rising.

System designers in the past had the luxury of creating code meant to live on a protected and secure proprietary network. Users were assumed to be benign. This is no longer the case. Just as many system designers and developers did not anticipate that the COBOL code they wrote in the ‘60s would live on to the new millennium (resulting in the Y2K problem), today’s system designers and developers may overlook the fact that their code will eventually be exposed to the Internet and all its dangers.

Many of the mobile code risks first discussed by McGraw and Felten in *Java Security* (1996) are only now becoming a threat. The movement toward web services and the “web-ification” of legacy systems using extensive middleware introduces extensibility risks. Modern software systems built on the .NET infrastructure and designed to interact in a distributed way on the Internet introduce important inherent risks that must be properly identified and managed.

By any measure, modern systems are more complex than legacy systems. Even the simplest measure of complexity (e.g., lines of code, or size) shows that modern systems are large and growing. Factoring in mobile code, distributed

---

systems and component-based middleware only makes the risks that much more apparent. From a reliability perspective, highly interconnected, distributed systems are extremely hard to design properly. Famous computer scientist Les Lamport once said, “A distributed system is one in which a computer that you did not even know existed can cause your computer to fail.” Software reliability in a network-rich environment requires careful risk management.

In theory, we could analyze and prove that a small software program was free of problems, but this task is impossible for even the simplest of today’s desktop systems, much less the enterprise-wide systems used by businesses or governments. A much more reasonable approach is to identify and manage software risks according to their anticipated business consequence. This allows reasonable technical decisions to be made squarely in the context of the business, avoiding both over- or under-spending.

**Software Quality Is a Process**

Software quality is a process, not a product. There is no substitute for working software quality as deeply into the software development process as possible, taking advantage of the engineering lessons software practitioners have learned over the years. Which particular software process is followed is not as important as the act of thinking about reliability, security, and performance as software is designed and built. Important software “ilities” such as security and reliability are attainable with today’s technology. The question is not whether good software is possible, but how good should software be for a given business use. By considering technical tradeoffs in terms of the business context, Cigital SQM solutions provide critical guidance to decision makers, resulting in higher quality software that is built both faster and cheaper.

Software engineering and security and reliability standards (such as the Common Criteria and ISO 9000) provide many useful tools that good SQM approaches can leverage. The key to building secure and reliable software lies in developing software under an iterative risk management regimen and applying tools and processes in a manner that is consistent with the business purpose of the software itself. An advanced SQM approach can answer the elusive “how” and “why” questions raised when a standard suggests certain technical activities.

**Measuring Software Quality**

Good software quality practices can help ensure that software behaves properly. The process of building better software is greatly enhanced by analyzing and measuring artifacts produced throughout the software process. Safety-critical and high-assurance system designers have always taken great pains to analyze and track software behavior. Designers of essential software for business must follow suit. This requires integrating Software Quality Management into the software engineering process starting as early as possible, and constantly iterating to improve the software being produced and the process being used to produce it.

*Gary McGraw, Ph.D. is the Chief Technical Officer at Cigital, Inc. He can be reached at gem@cigital.com.*