Software security assurance a matter of global significance: product life cycle propositions

Doreen (Dee) Sams
Georgia College and State University

Phil Sams
Sr. Load & Performance Engineering Consultant
Milledgeville, Georgia USA

ABSTRACT

The global business world of the 21st century has seen catastrophic financial losses to companies brought on by security breeches. The business world today dictates the urgency and importance of frontend life cycle software security development. The objective of this research is to examine factors impacting security-oriented software development from a holistic product life cycle approach and the outcomes. This conceptual piece puts forth measurable propositions based on literature research and industry expertise as a first step to empirically examining the benefits and cost of a holistic approach to software security development. Findings from the literature and industry expertise clearly indicate the need for early product life cycle development of software security.

Key Words: Software Security, Functional Testing, Software Development Product Life Cycle, Financial Impact

INTRODUCTION

This propositional paper is a first step in the process of empirically addressing an immediate and growing strategic business decision as to the benefits of computer software security testing beyond the traditional quality control testing involving functional (i.e., assurance that software functions as it is intended) and load-performance testing to software security testing. The paper focuses primary on the software security testing (formerly known as application security) decision as to cost (i.e., short run and long run) versus benefits within the software development life cycle (SDLC).

Technical Definitions

Terminology is important in the understanding of software development and testing. Therefore, the following definitions are offered. Quality Assurance (QA) is the prevention of defects. Quality Control (QC) is the detection (testing) and removal of software defects. Functional testing is the software verification process that determines correct or proper application behavior for only one user. Regression testing is verifying that what was working in a previous application release still works in subsequent releases. Load-performance testing is the process of testing an application to determine if it can perform properly with multiple concurrent users, possibly thousands. Security testing goes much deeper than traditional functional or regression testing. For the purpose of this study, software risk is defined as the combination of the likelihood of a defect occurring and the potential impact of the occurrence. Software security testing verifies correct software behavior in the presence of a malicious attack. Vulnerability is a software error that an attacker can exploit. Therefore, it is important to understand that "software security is not security software" (McGraw, 2004, p. 33).

Background

In the past, the role of quality control (QC) testing has, by necessity, focused on testing of application functional and perhaps load and performance testing and not globally on software security. Software security encompasses, but is not limited to software security, security management, planning and operations security, physical security, network, and Internet security. Software security testing involves the person(s) *who* should do the *what* security testing functions.

In today's business environment, functional testing of WEB or ecommerce applications or load-performance testing alone is not sufficient (Gallagher, Jeffries, and Landauer 2006). Literature (e.g. Gallagher, et al., 2006) reveals, application functional, regression, and load-performance testing has become more generally accepted as a necessity in the SDLC. On the other hand, application security testing, a vital element of strategic business operations is possibly today's most overlooked aspect of software security and has not yet been given due-diligence.

The tipping point that ushers in the necessity for software security testing beyond the traditional functional and load-performance testing is the additional risk brought on by the globalization and internationalization of markets. The rapid integration of national economies into the international economy comes in part through the spread of technology (Bhagwati, 2007). Companies whether national or multinational, are affected by rapid advances in global technology that has created great opportunities for expansion into new markets and increased revenues. However, along with the benefits of globalization, come negative and sometime unforeseen consequences such as the invasion and theft of business critical proprietary software and data of company secrets, customers' personal data, and possibly the destruction of a company's ability to perform competitively in the global marketplace. Risks have increased through higher levels of integration among business systems such as Customer Relationship Management (CRM), etc.; thus, creating a need for even stronger application security measures to circumvent and prevent extensive financial losses. Therefore, the role of risk-based software security testing (i.e., the probing of risk previously identified through risk analysis) is changing from optional to mandatory.

In order to remain financially sustainable in a globalized economy, software security must be seen as more than a tool; it must be viewed from a systems perspective. Taking a systems perspective, application security is part of a discipline integrated in a total quality management process involving test of the entire software system and not merely parts of the system. Costs associated with poor or inadequate software security have grown rapidly; thus, companies are beginning to invest in security training, automation testing tools, and various associated quality assurance (QA) and quality control (QC) methodologies to circumvent cascading costs that typically occur over the software development process. Cascading costs include cost for: requirements, design, implementation, testing, and production defects. This study proposes that prevention of cascading costs by holistically engaging in global (i.e., systems) security of the software system necessitates that as each new product is developed in the research and development stage of the product life cycle that testing should begin early in the research and development (R&D) phases.

Therefore, this study examines the role of software security testing as a cost reduction of enterprise applications, frontend resource application, and risk reduction methodology in the prevention of the potential for catastrophic financial impact on the company developing the software, and the user of the software application product. This study is a first step, in that, it puts forth measurable propositions that can be addressed through mixed methodologies such as surveys, in-depth interviews and focus groups of software test engineers, managers, and clients across a wide-variety of companies and industries.

LITERATURE REVIEW AND PROPOSITIONS

Sustainability, by its very nature, involves using resources of the company in such a manner that the company remains financial sustainable over time. Moreover, to reduce security risk, development of risk management must begin in the first stage of the SDLC.

A financially sustainable company wisely plans the use of limited-resources (i.e., materials and human capital); thus, to be sustainable means planning for security of software must begin in the research and development (R&D) stage of a product's life cycle. The product life cycle as defined by Kotler and Armstrong (2011), begins in the research and development stage (i.e., product development) of a product's life. This is a time when expenditures may be high and sales are zero. From this stage, the product enters the introductory stage, in which sales growth is slow, marketing costs are high, and expenses heavy. From there the product enters its growth stage and then there is a maturity period that leads into the final stage known as the decline stage (Kotler and Armstrong 2011). However, not all products follow this typical life cycle and many go into decline rapidly for various reasons. These products are financial failures for the company. On this premise, benefits gained through early defect prevention enabled by early automated testing in the R&D stage of the product life cycle are expected to significantly outweigh the financial costs involved in fixing the problems later, loss of business, and/or negative word of mouth and possible lawsuits.

Through the vulnerability of the product comes vulnerability of the company. Attacks may occur at any stage in the product's life cycle; however, it is proposed that protection must begin as each product idea enters the development stage of its life cycle. It is here where cascading costs can be circumvented by implementation QA and QC best practices. Once the product is introduced into the market, the risk is then shared between the development company and the company implementing the product (i.e., software). However, the risk to the development company may be the greatest. Customer satisfaction with the quality of the product is measured in performance (i.e., ability to perform its functions) and conformance (i.e., freedom from defects), while high quality also involves consistency in the product's delivery of benefits that meets the customer's expectations). If the product meets or exceeds performance and conformance, but does not function at the level of the customer's expectations consistently, the customer is negatively disconfirmed. Hopefully, the customer will complain and not merely switch providers. However, the perception of the likelihood of a successful complaint means that the complaint must result in a corrected or changed situation. Perceived likelihood of success comes from the customer's perception of the company's fairness of the procedures and policies in arriving at a remedy, the remedy itself, and the perception that the treatment has been delivered in an acceptable manner (Homans 1961, Lind and Tyler 1988). Thus, many companies' software development involves augmenting the product with product support and after sale service, yet this may not be enough if the overall costs of the defects to the buyer are too high or too often (e.g. when defects cause excessive loss in time, productivity, and money).

Tax, Brown, and Chandrashekaran (1998) found that if complainants believe that they have not been successful in gaining resolution to the problem, then they have a propensity to engage in negative word of mouth or exit the relationship. If the problems persist and/or the company does not respond quickly and in proportion to the problem, the customer may abandon the product and the company, which results in a financial loss to the company. Moreover, even greater damage comes from negative word of mouth advertising from the dissatisfied customers. In today's electronic age, negative word of mouth spreads at Internet speed and the result to the company can be catastrophic. Thus, engaging in holistic (i.e., systems) software security testing in the developmental stage of the product by identifying risks to the company from what may be perceived as the smallest threat, gives the company the potential by which it can avoid immediate and long-term financial risks.

A risk analysis for the development company, by its nature, must assess risk costs based on the actual risk, the size of the risk (as to the extent of cascading affects), its immediate and long-term impact on the company's sustainability, prevention costs (i.e., personnel, software packages, etc.) verses benefits to the company in the short and long run. In the short run, upfront costs come from the purchase of automated software testing tools ranging in cost from $5,000 - $250,000+ for tools, plus typically 20% for annual maintenance. Additionally other expenses typically include a set amount of tool specific training factored into initial costs. Regarding human capital costs, depending on where software quality assurance testers are located, salaries fall in a range of $35,000 - $60,ooo annually for full time manual tester. Automated software testing is a highly specialized area within the computer science field and requires extensive tools training as well as a minimum of a four-year computer science degree. Therefore, companies often hire automated software consultants. Consultants are used for short-run initiatives and a company may pay an automated testing software engineer anywhere from $60,000 to $150,000 annually plus travel and expenses. These consultants' contracts typically run from three months to a year depending on the project and the company perceived needs. The consultants are often contracted for companies that have short-term needs such as developing an automated testing framework, or load and performance testing. The variation in salary is based on the software engineer's expertise with automated testing tools, experience in the field, educational degrees, and the level of risk associated with the company's product (e.g., medical supply companies such as Baxter Health Care must, by the nature of their product and the level of risk to the client and the company, hire extremely talented, competent, and experience automated test engineers).

Therefore, frontend quality assurance (QA) provides significant value (i.e., cost verses benefit) with regard to reducing defects and costs of all software, including security testing. Study after study, such as the study by Pressman (2005) "Software Engineering: A Practitioner's Approach," have shown that as a defect progresses from requirements to the next phase of the software development life cycle (SDLC), the approximate cost of fixing a defect increases by a factor of ten at each phase of the SDLC. According to National Institute of Standards and Technology (NIST), 80% of costs in the development stage are spent on finding and fixing defects. Further, the NIST suggests that a preemptive approach of building security and compliance into the frontend product reduces vulnerability and costs less in the long run (Anonymous 2009). In other words, by the time a defect makes it through requirements, design, development, testing, and to production, the cost of fixing the defect increases exponentially. Early stage (i.e., R&D) specification of software security requirements, designing and coding are likely to provide a substantial cost savings by preventing security bugs rather than bug fixes after the software application is in production. Although planning for security risks begins in the development phase, it must be assessed and controlled throughout the products life cycle. Once the quality control (QC) testing phase is entered, having automated tests ready to go, including automated software security test cases, dramatically improves the ROI garnered from the testing phase. If automated tests are developed during the SDLC as well as with each software release, the test team will have a significant inventory of automated tests, both functional and software security. The larger the inventory of automated tests the more efficient and effective the test phase will be in addition to securing a higher ROI over manual testing. Hence the following is proposed:

P_{1a}: Inclusion of frontend software security features reduces total cost of enterprise applications.

P_{1b}: Testing of frontend software security features reduces total cost of enterprise applications.

Proposed Implementation Plan

A security issue, in the QA/QC world, is viewed as a defect, but one with a very high risk. The tester should use a risk-based approach to software security development with architecture in mind, and they must identify risks to the application and focus on areas of code that may be vulnerable to attack.

For some companies, in addition to frontend costs to prevent incidents, the perception of how significant a security risk is for that company plays a key role in whether frontend QA is implemented. Therefore, typically, one of the first tasks to be completed in the software development process is a risk analysis. Likewise, in testing software security, a high priority is given to performing an effective analysis of software security risks. It is a generally accepted fact that it is impossible to test everything in an application, including security. To identify and test software security, a test engineer must add to their current testing knowledge the mindset of a 'hacker' and move beyond the conventional QA/QC testing mindset. The software security tester must: 1) apply conventional QA/QC testing methodology, 2) thoroughly understand what is being tested, 3) think outside the box (i.e., maliciously about the target software or module), 4) attack the target software by applying malicious ideas, methods, and data, and 5) stay informed about potential threats that might affect the testing target (Gallagher et al., 2006). To address strategic risk through conventional analysis allows the analyst to prioritize what risks need to be addressed according to their risk level. However, a company must obtain and maintain both conventional security and quality assurance risk assessment measures while engaging in unconventional thinking in order to secure the company's financial sustainability.

One effective means for visualizing the likelihood and the potential impact of a defect is through a table format with defect likelihood on one axis (e.g., rows) and potential impact of the defect on the other axis (e.g., columns). Characteristics of the risk can be prioritized as to impact on systems or critical software and recovery using a weighted risk. It is necessary to define the weighs as to the meaning of critical, high, medium and low for each risk. See Table 1 (all tables are in the Appendix).

One security model utilized to assess risk, given credence by Certified Information Systems Security Professionals.Com (Cissp.com 2010) and the computer security community at large is the CIA model which is an acronym for: 1) confidentiality – who has access to your software and data, 2) integrity – is the software functioning accurately and is the data accurate, and 3) availability – can authorized users access the software get to the application and data?

In order to create a viable software security test plan, a list of all threats to critical functionality should be compiled. The list of entry points should include all possible means as to how the data is used and how it might be used maliciously to cause undesirable results and threats to critical functionality (Wright, 1994).

Once risks have been identified, the next logical step is the planning stage. Planning is a key to success in any software endeavor. In the book, Hunting Security Bugs (Gallagher, Jeffries, and Landauer 2006), the authors provide a *threat model,* to assist practitioners in planning the software security-testing task. The *threat model* has three key parts: 1) data flow diagrams (DFD), 2) enumeration of entry and exit points, and 3) enumeration of potential threats.

According to the threat model, data flow diagrams (DFD's) provide testers with a clear understanding of how, where, and what data flows between software components. DFD's can be used for the entire application, or perhaps just specific functionality. Data objects of particular concern are: personal information, account numbers, passwords, and data from anonymous sources. Additionally, DFD's highlight the functional areas that require specific sensitive information, or perhaps generate explicit sensitive information (Gallagher et al., 2006).

Entry points are susceptible to malicious data entering the application, and as such, should be thoroughly tested (e.g. negative testing). Malicious data may be used to 'unlock' a module, function, or method allowing an unauthorized user to further misuse the application and or its data. Typical security concerns of this type are: 1) controlled access (customers and employee business), 2) confidentiality protection (disclosure of sensitive information), 3) integrity of data (protection from unauthorized modification), 4) non-repudiation of data (original data or audit controls), and 5 monitoring and auditing security process and procedures (Gallagher et al., 2006). Exit points of particular interest include: 1) business objects (especially those regarding security), 2) organization information (user roles, etc.), 3) process components (including data structures), and 4) partners (and relationships) (Gallagher et al., 2006).

Once risks are assessed and identified, utilizing a gumball approach for software application security is proposed as a best practice and preferred over an approach of putting all of the security eggs in one basket (e.g. firewalls). The gumball model uses layers of security, much like an onion. Due to the risk involved, the Transportation Security Administration (TSA) uses this methodology to implement several security screenings and checks (J. Whitney Bunting College of Business 2007). By using the gumball implementation methodology, the malicious user is unlikely to know how many layers he or she must go through to get to what they want, and may be more inclined to move on to another easier target. One important thing to remember is that software security does not stop at the borders; therefore, the deeper the hacker must go to get to their desired goal, the less likely they will be successful.

For management another strategic security decision is whether to implement computer security at the infrastructure level or with security code in each application. To ensure success in securing computers and applications, strategic decisions should once again take a systems approach by considering the roles of people, processes, and technology (Nagaratnam, Nadalin, Hondo, McIntosh, and Austel, 2005). Intuitively, software security testing can provide a vital role and feedback in the generally accepted (Security) Process Engineering model of: Plan: 1) plan – create security processes (QC the processes before implementation), 2) establish metrics (QA, provides feedback on what metrics will work well and which may not), and Do: 1) implement the security plan (QC tests software security per implementation), and Check: 1) measure and monitor (QC, software security defects and issues), and lastly, Act: 1) review and improve software security (QA provides statistics on software security test results), and 2) improve continuously (QC provides information on what was good, bad, and ugly).

Based on 20 plus years of experience in the quality assurance/quality control (QA/QC) testing field and on existing data, it is proposed that the aforementioned software security categories are vital to the success of a company's business performance and should be viewed holistically to provide a comprehensive software security solution. Based on research as to the above-recommended frontend resources, the following is proposed.

P_2: Incorporating software security features into the research and development phase results in lower costs and less security risks than developing software security features later in the product life cycle.

It has been said that those who do not remember history are doomed to repeat it. Some companies that have fallen victim because of the absence of software security include: Volkswagen (loss of $260 million to an insider scam of phony currency-exchange transactions), Bank of New York ($32 billion lost due to a processing error), hackers victimized Southwestern Bell and other companies, Southwestern Bell alone reportedly spent $370,000 to repair programs and add software security. The fate of these companies should be "red flags" for others. In spite of the losses to the companies that use the product, it is obvious from previous research that the catastrophic risk belongs to both the company that developed the software to the company that uses the software. However, the greater loss and risk fall on the developing company. Nevertheless, the risk exits for both. Therefore, the following are proposed:

P_{3a}: Inclusion of software security features into the enterprise application reduces the risk of catastrophic financial impact on the company that develops the software.

P_{3b}: Inclusion of software security features into the enterprise application reduces the risk of catastrophic financial impact on the client.

CONCLUSION AND MANAGERIAL IMPLICATIONS

Although research exists in automated software testing, the benefits of a systems-testing approach, relative to early production development and life cycle testing have received little attention by academics or practitioners. The cost of ignoring a systems approach to life cycle security testing can be ruinous to the developing company and/or the customer. It is further recognized that software application security is one of the greatest concerns of many software organizations and yet one of the least understood and implemented testing tasks. Software security testing is very different from functional or load-performance testing and as such requires a domain of expertise far beyond conventional testing methods and practices (Gallagher et al., 2006). However, the benefits of the inclusion of software security testing relative to cost savings through frontend of mainstream of the QA/QC testing phase of the SDLC is of foremost importance in averting security risks across many types of the software applications. Nevertheless, building security into the frontend does not mean that security issues can be forgotten throughout the product life cycle. However, this does have the potential to reduce the cost associated with application software security, as well as traditional software vulnerabilities.

To fully investigate the propositions brought forth in this paper, interviews should be conducted across industries, companies, managers, and software engineers. Findings from these interviews could be used to create an appropriate survey instrument to capture a larger sample. The full investigation of the propositions is expected to add value to strategic management decision making by revealing the extent of the benefits of life cycle testing.

REFERENCES

Bhagwati, J. (2007). In defense of globalization: With a new afterworld. New York: Oxford University Press.

Cissp (2010). Home page. Retrieved September 10, 2010, from http://www.cissp.com.

Gallagher, T., Jeffries, B., & Laudauer, L. (2006). Hunting security bugs. Washington: Microsoft Press.

Homans, G. (1961). Social behavior: Its elementary forms. New York: Harcourt, Brace, and World.

J. Whitney Bunting College of Business, Georgia College and State University. (2007). Workshop for computer and Information Security. (September): Macon, Georgia.

Lind, E., Tyler, A., & Tyler, T. (1988). The social psychology of procedural justice. New York: Plenum.

Kotler, P. & Armstrong, G. (2011). Marketing an introduction. 10th Ed. Massachusetts: Prentice Hall.

McGraw, G. (2004). Software security testing. *The IEEE Computer Security Society*: 32-36.

Nagaratnam, N., Nadalin, A., Hondo, M., McIntosh, M., & Austel, P. (2005). Business-driven application security: From modeling to managing secure applications. *IBM Systems Journal,* 44, 847-867.
Newswire (2010) IBM acquires Ounce Labs, Inc.: new capabilities extend IBM's application security and compliance offerings – help lower risk and costs of software delivery. (2009, July 28). PR Newswire. Retrieved August 25, 2010, from http://www-03.ibm.com/press/us/en/pressrelease/27971.wss.
Pressman, R. (2005), Software Engineering: A Practitioner's Approach. McGraw Hill: New York, NY.
Tax, S., Brown, S., & Chandrashekaran, M. (1998). Customer evaluations of service complaint experiences: Implications for relationship marketing. *Journal of Marketing*, 62, 60-76.
Wright, M. (1994). Protecting information: Effective security controls. *Review of Business,* 6, 24-28.

APPENDIX

Table 1 Risk: Likelihood and Impact of Defect

	Impact		
Likelihood	Catastrophic	Damaging	Annoyance
Unavoidable	C	H	M
Likely	C	H	L
Rare	H	M	L

Weighted Risk: C = Critical, H = High, M = Medium, or L = Low