Software Assurance: A Kaleidoscope of Perspectives

Mano Paul, CSSLP, CISSP, AMBCI, MCAD, MCSD, Network+, ECSA

Introduction

In this day and age when software is rife with vulnerabilities, as is evident in full disclosure lists and hacking incidence reports, security in the software lifecycle can simply no longer remain on the sidelines. Software security breaches and data loss have resulted in devastating fines, irreparable reputation damage, and in some cases have jeopardized the very survival of many companies and organizations. And while writing secure code and passing hack-resilient tests are important elements of software assurance, they represent only a starting point, a subset of the holistic umbrella term "Software Assurance".

Frost & Sullivan research from two (ISC)$^{2®}$ studies has found there are two primary conditions that create information security vulnerabilities in enterprise software applications:

1. Inexperienced developers writing code
2. Influencers not understanding information security issues as they pertain to the Software Development Life Cycle (SDLC)

Influencers naturally have differing points of view and priorities. But individual priorities must give way to the overall success of the development of secure software. Budgets and design specifications must be developed with security in mind. Poor design invariably results in a product with inherent security flaws, and budget limitations lead to overruns and higher maintenance and enhancement costs.

Prior to the software's market release, influencers have differing levels of power over the Software Development Life Cycle (SDLC), from the conceptual stage to the development and testing stages, and everything in between. To be "truly secure," one must address the elements of software security through the entire lifecycle, from initiation to sunset, or, in other words, from envisioning and planning to disposal.

(ISC)^2's whitepaper, *The Need for Secure Software* addresses the "Why" of designing, developing, and deploying secure software. It delves into the drivers of software assurance and the importance of data security. It covers the policy, process, and people aspects of software assurance. This paper will address the "What" of secure software. What does it take to design, develop, and deploy secure software?

What is Secure Software?

First and foremost, it's important to understand that secure software does not mean zero-defect software or software that is 100% hack-resilient with no vulnerabilities. While from a security standpoint this would be an ideal situation for all software, it is purely utopian. Such software does not exist. All software is prone to attack unless it is in a non-operational inaccessible state.

Secure software is software designed with security in mind, developed with security controls, and deployed in a secure state. While it might have the potential of being breached, the repercussions of a breach are greatly diminished. Secure design and architecture, secure development with security controls built in by default, and secure deployment or release, all work together to minimize the impact of a software vulnerability that gets exploited. In other words, secure software is about mitigating or controlling the risk of software vulnerabilities.

A Kaleidoscope of Perspectives

Peering out into the landscape of software development, it's clear that there is no magic silver bullet in designing, developing, and deploying secure software. In fact, with the advent of new technologies and the increasing rate at which technology is changing, software developed and deployed today in a secure state may no longer be secure in the future. And merely fixing software vulnerabilities with a patch-and-release cycle, as is predominantly the case today, does little to get to the root of the problem.

Various actors, factors, and perspectives need to be considered. Software development should be viewed as a set of patterns that are reflected in a kaleidoscope depending on the angle of viewing. Software development is non-static and reflects different perspectives depending on the vantage points of the stakeholders involved.

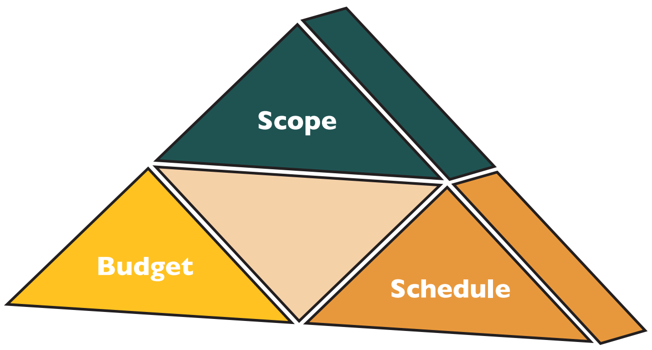
These varied perspectives need to be considered in design, development, and deployment of software, and software assurance should be an amalgamation of them. The typical perspectives in software development can be categorized as follows:

i. Organizational Stakeholders
ii. Business vs. Information Technology
iii. Information Security and Risk
iv. Software Development Models
v. Software Development Environment

i. Organizational Stakeholders

The nature of enterprise software development requires collaboration across various teams made up of personnel with different roles. Some of these roles include the client, business analysts, requirements analysts, product managers, project managers, software engineers, designers, architects, development managers, developers (coders), testers, and operations personnel. Though operations personnel are not directly involved in building the software, they play a vital role in ensuring that the deployed software is operated securely and remains secure. In some cases, executive management is also part of the software development project stakeholder list. While all stakeholders should participate in building secure software, it's important to note that executive management plays a pivotal role in software assurance. The success of a software assurance program within an organization is directly proportional to the support from executive management. This top-down support has an indelible influence on all stakeholders, from the client to the coder, to develop software securely.

Figure 1. Scope, Schedule, and Budget SDLC Iron Triangle



On the surface, it may seem that the officers and executives of the corporation may have little or nothing to do with software assurance. However, following various fraud and data breach incidents, a barrage of regulations and compliance initiatives has been enforced, including Sarbanes-Oxley (SOX) and the Gramm Leach Bliley Act (GLBA), to name a few. These regulations hold executive management responsible for software insecurity. Individuals in the executive echelons are ultimately responsible for protecting customer trust and therefore are indirectly influential in software assurance. In security circles, ROI which conventionally stood for Return on Investment has, with grim humor, been replaced with "Risk of Incarceration".

When developing software securely, all stakeholders should have an appropriate level of participation. The mantra, "Everyone is responsible for Software Assurance" is not an overstatement by any measure.

ii. Business vs. Information Technology

As mentioned earlier, research indicates that one of the main reasons security vulnerabilities find their way into enterprise software applications is because the influencers don't understand security issues as they pertain to the SDLC. One group of these influencers is the client/customer (if external) or the business unit heads (if internal), collectively called the "business" for whom, the information technology (IT) teams are developing the software. The business specifies the functional requirements of the software but seldom specifies the security requirements. Constraints in scope, schedule, and budget budget, as shown in Figure 1, are often the reasons why security requirements are left out. If the software development project's scope, schedule (time), and budget are too rigidly defined, then there's no room left for the team to maneuver, and failure is inevitable (because "something's gotta give"[a]). Unfortunately what's typically sacrificed are the elements of software security.

"While there have been many studies detailing out the cost associated with fixing vulnerabilities in the production environment versus finding them further upstream; they often do not include many of the interim expenses we have found to be significant in the remediation process. This price often includes operational work and support to create patches and mitigating controls while the root causes are to be addressed in the software."

Ed Bellis,
CISO and Vice President, Orbitz Worldwide

Another reason for leaving out security requirements in software is that the client or business units may not know how to articulate the security requirements adequately enough for the IT teams to incorporate them as they develop the software. Additionally, in some cases, the IT teams are not trained to ask for security requirements or translate the functional requirements into security requirements. It is however imperative that for software to be secure, client and business requirements should be understood by the IT teams. Security requirements in addition to the functional requirements should be requested or generated. IT should understand the risk and business units should understand security, at least at a high level. If constraints are imposed on the software development projects, and necessary risk mitigation measures are not made available for whatever reasons, the client or business unit should be made aware of possible vulnerabilities that may exist in the software being released and be willing to accept or transfer the risk.

Software developed should not only be business-aware but also secure from a technical perspective. At the bare minimum, common technical controls addressing confidentiality (who can see the information), integrity (who can modify information), availability (when the information is, or is not, accessible), authentication (who is making the request), authorization (the rights and privileges of the requestor), and auditing (historical evidence), should all be built in. Anything less than bare minimum security in software, increasing the risks of a breach and the serious repercussions thereof, is the equivalent of driving a car without seat belts thus increasing the risk of a fatality in the event of an accident.[b] Operationally, the software should run with the least privilege and stay secure. Fail-safe controls should be built in. Privileged and administrator-level access should be controlled and audited.

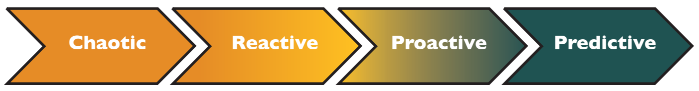
Both the business and IT teams should share the risk. This will lead to an increase in software security, and eventually a heightened, and more mature organization with software assurance.

iii. Information Security & Risk – A Balancing Act

In most cases the business units express and understand information security in terms of risk, and not technical information security controls. Meanwhile the technical IT teams express information security in terms of technical vulnerabilities and controls, and not in terms of risk. Naturally this difference serves to widen even further the already-existing communication gap between the business units and the IT teams.

Information security and risk are two sides of the software assurance coin and, irrespective of what side the coin lands on when flipped, both need to be equally and effectively addressed. An organization's maturity pertaining to risk, as presented in the 2007 Computer Security Institute (CSI) conference, will fall somewhere in the Risk Spectrum[c] from a chaotic to a predictive state, as shown in Figure 2.

Figure 2 - Risk Spectrum (Chaotic – Predictive)



In the chaotic state, risk is not managed in a structured manner and even infantile components of risk management don't exist. In the reactive state, risk is managed on an ad-hoc basis, typically as a result of an incident or discovery of a potential exposure or vulnerability. In the proactive state, risk is addressed prior to any incident or discovery of potential exposures or vulnerabilities. In the predictive state, not only is risk addressed proactively, but projections of possible exposures in the future are made. Most organizations today fall somewhere between the chaotic and reactive states of risk, with some falling between the reactive and proactive states. But as software assurance gains more momentum, with user-awareness, education and certifications, we can expect to see organizations fall more between the proactive and predictive states of risk.

As software assurance gains more momentum, with user-awareness, education and certifications, we can expect to see organizations fall more between the proactive and predictive states of risk.

To address risk means that security in software is part of the equation, and to address security in software means that risk is being addressed, accepted, mitigated, or transferred – and never ignored. Addressing one without the other should never be an option. Software assurance is about balancing information security and risk, as shown in Figure 3.

Figure 3. Software Assurance – A Balancing Act

iv. Software Development Models

SDLC is an acronym used for Systems Development Life Cycle or Software Development Life Cycle. Either way, it's a process employing various kinds of expertise and technology, usually comprised of a phased approach, and sometimes with overlapping phases.

There are several SDLC models that are prevalent in today's enterprise including:

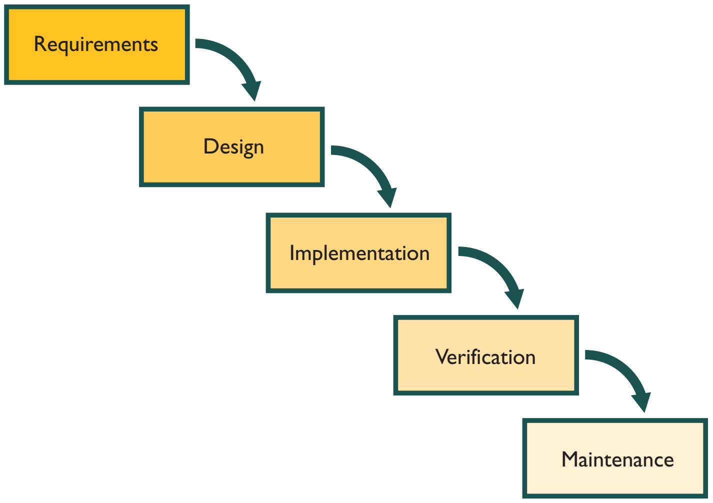
- The Waterfall model
- Iterative (Prototyping) model
- Spiral model
- Extreme Programming (XP) models of Agile methodology

The traditional, structured Waterfall model is characterized by a linear, sequential process in which software being developed flows downward like a waterfall, through phases with fixed specifications. Royce's original waterfall model (1970), incorporates the following phases, to be followed in order:

1. Requirements specification
2. Design
3. Construction (a.k.a. implementation or coding)
4. Integration
5. Testing and debugging (a.k.a. verification)
6. Installation
7. Maintenance

Once a phase is completed, the software development process moves on to the next stage, as shown in figure 4. This Waterfall model is used by large development organizations especially for large software projects because it brings structure by phases to the software development process. The National Institute of Standards and Technology (NIST) Special Publication 800-64 REV 1[d], covering Security Considerations in the Information Systems Development Life Cycle, breaks the linear Waterfall SDLC model into five generic phases: initiation, acquisition/development, implementation/assessment, operations/maintenance, and sunset.

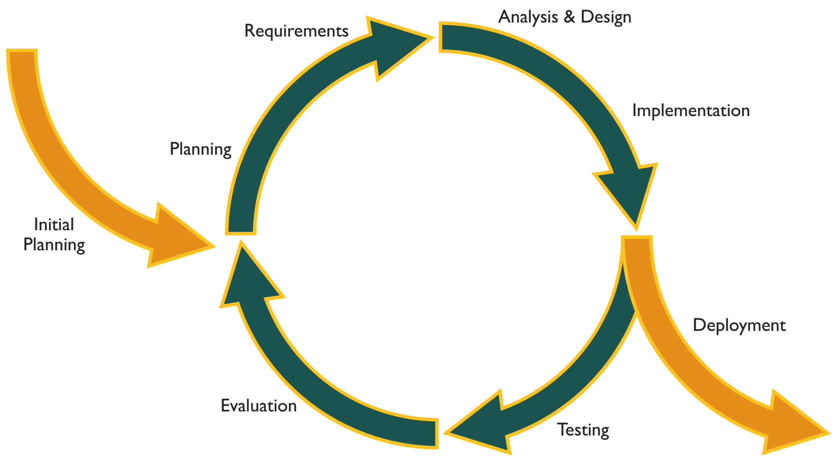
**Figure 4.
Waterfall Software Development Model**



Today, there are various modified waterfall models that may include different phases with slight or major variations.

In the Iterative (or prototyping) model, the software development project is broken into smaller versions and developed incrementally, as shown in figure 5, as the team learns from one version of the software to the next. This allows the development effort to be aligned with the business requirements, uncovering any important issues early in the project and thereby avoiding disastrous faulty assumptions. It is also commonly referred to as the prototyping model in which each version is a prototype of the final release to manufacturing (RTM) version.

Figure 5. Iterative Software Development Model



The Spiral model, as shown in figure 6, is a software development model that has elements of both the waterfall model and the prototyping model, generally for larger projects.

The Agile methodology or Extreme Programming (XP) model is built on the foundation of iterative development and aims at minimizing software development project failure rates by developing the software in rapid iterations (called timeboxes). It uses feedback that is driven by regular tests and releases of the evolving software as its primary control mechanism, as shown in figure 7, instead of planning as in the case of the spiral model. The Agile methodology or XP model is also referred to as the "people-centric" model of programming and is more useful for smaller projects.

Figure 6. Spiral Model

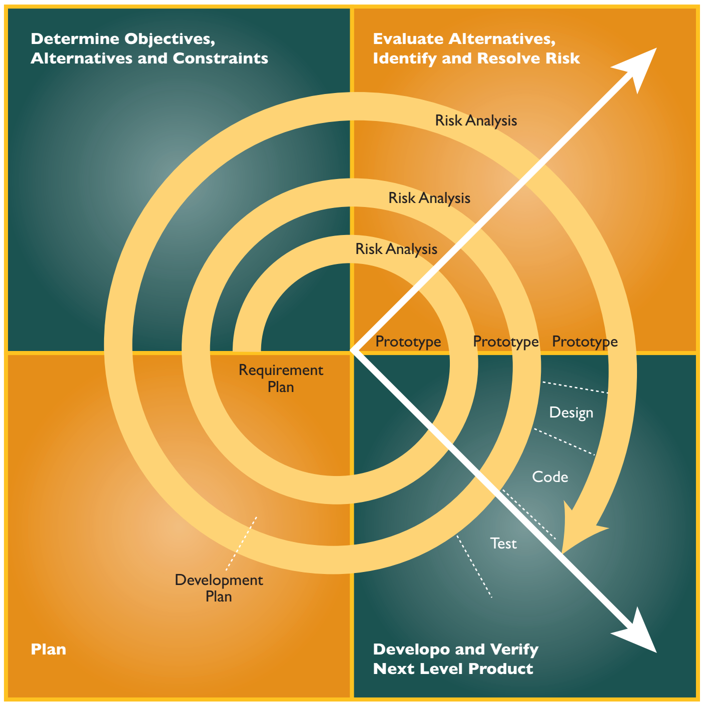

Determine Objectives, Alternatives and Constraints
Evaluate Alternatives, Identify and Resolve Risk
Risk Analysis
Risk Analysis
Risk Analysis
Prototype
Prototype
Prototype
Requirement Plan
Design
Code
Test
Development Plan
Plan
Developo and Verify Next Level Product

Figure 7. Extreme Programming Model

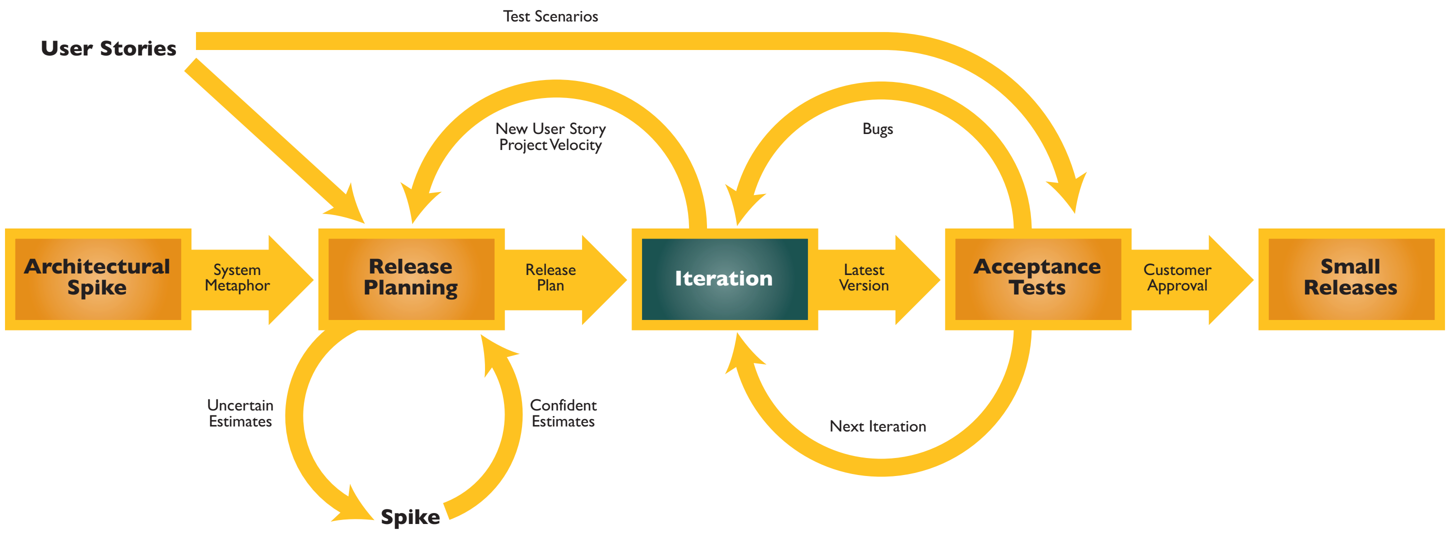


Figure 8. SCRUM Methodology

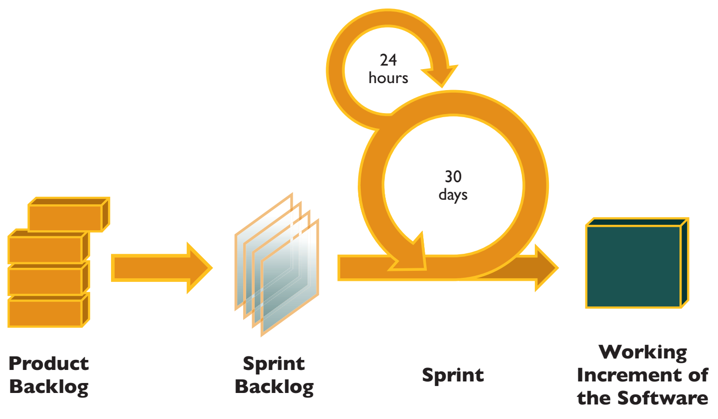


Figure 9. Security Integral to the

Phase	Step#	Step
Envisioning	1	Identity Threat/Risk Vector(s)
Planning	2	Profile Software
	3	Threat/Risk Modeling
	4	Generate Security & Risk Requirements
Developing	5	Control Check
Release	6	Handle Threat/Risk
Stabilization	7	Learn and Educate

Another very popular and widely used recent Agile development methodology is the Scrum programming approach. Scrum approach calls for 30-day release cycles to allow the requirements to be changed on the fly, if necessary. In Scrum methodology, the software is kept in a constant state of readiness for release, as shown in figure 8. The participants in SCRUM have pre-defined roles, which are of two types dependent on their level of commitment viz. Pig Roles (Committed, whose bacon is on the line) and Chicken Roles (Participating). Pig roles include Scrum Master (Like the project Manager), the Product Owner who represents the stakeholders and is the voice of the customer and the Team (the developers). The team size is usually 5-9 for increase communication. Chicken roles include Users (those who will use the software being developed), the Stakeholders (the customer or vendor) and Managers. A prioritized list of high level requirements is first developed which is known as a Product Backlog. The time allowed (usually about 30 days) that is allowed for development of the product backlog is called a Sprint The list of tasks to be completed during a Sprint is called the Sprint Backlog. A daily progress for a Sprint is recorded for review in the artifact known as the Burn Down Chart.

In most cases, the most conducive model for enterprise software development has been a combination of two or more of these models. It's important, however, to realize that no model, or combination of models, can create inherently secure software. For software to be securely designed, developed, and deployed, a minimum set of security tasks needs to be effectively incorporated in the system development process, and the points of building security into the SDLC model should be identified. Figure 9 depicts a minimum set of security tasks that needs to be an integral part of the generic SDLC through all the phases of a software development project.

v. Software Development Environment

Gone are the days when software development was contained within an organization's perimeter, and even within the borders of a country. With the rise in access to inexpensive labor, and the competitive advantages such labor produces, many organizations jumped on the bandwagon of outsourcing, sending their software development projects to countries in the emerging/establishing marketplaces of Eastern Europe, Russia, India, and China. In the interest of business operations, demarcating network access devices such as the firewalls and demilitarized zones that separated the outside from the organizational assets started to slowly disappear, and the world became one big, global development shop. In the new, seemingly perimeter-less global development world, designing, developing, and deploying secure software is a challenge, to say the least. No longer can software be hidden behind the defenses of a firewall. To further exacerbate the vanishing perimeter state of affairs, companies in countries that were outsourced to are, in turn, outsourcing to countries with even lower labor costs than themselves. Tracing accountability in such scenarios can be exceedingly difficult. It is crucial that global development methodologies and motivations be factored into the software being developed, and that building security controls in such a state of affairs not be neglected.

The First Line of Defense – Qualified Personnel (Aware, Skilled and Certified)

Contrary to popular thinking that education and awareness are essential elements of a "lessons learned" effort that occur after software has been deployed/released, the Microsoft Press book, The Security Development Lifecycle[e] lists Education and Awareness as Stage 0 (zero) of an SDLC project. That is to say that education and awareness precede even the Project Inception phase of a project. The Security Development Lifecycle attributes education and awareness (along with executive support) as one of the two critical success factors in reducing the vulnerabilities in Microsoft software.

Microsoft hit the "why-do-we-still-release-vulnerable-software" nail on its head. If the stakeholders in the software development process are not aware of common security tenets and threats, and are not skilled to incorporate security controls into the software, any attempt of software assurance to design, develop, and deploy software securely is futile. While a secure software development organization is one that has its personnel aware and educated on software security, a "mature" software development organization is one that, in addition to having its personnel made aware and educated, will also have them qualified by certifying their level of understanding and demonstrable expertise.

Conclusion: What Next?

With insecure software rampant in today's business environment, and in light of mounting regulatory and compliance requirements, building secure software can no longer be thought of as an activity on the fringe. Building secure software is a result of all the stakeholders having the appropriate levels of participation, and a security mindset in the design, development, and deployment of the software.

Software assurance has a kaleidoscope of perspectives that need to be factored into the secure software lifecycle. Software Assurance stretches from the boardroom to the builder, from the client/customer to the coder. Effective software assurance includes non-technical, non-developer roles as well from the business, management (people, product, and project), and operations.

The first line of defense in software assurance is qualified and educated personnel.

Everyone is responsible for Software Assurance. Both information security and risk are to be adequately addressed and software assurance is about balancing the two. Additionally, software assurance is about a minimal set of security tasks made integral to the SDLC, irrespective of the software development model. Another vantage point is the changing software development environment. Today's global environment, characterized by a vanishing perimeter, with the outsourced company often outsourcing and development often disconnected, means that ignoring software assurance will cause potential breaches to be inevitably realized.

The first line of defense in software assurance is qualified and educated personnel. These are the individuals who have the necessary awareness and are trained with the necessary skills to design, develop, and deploy secure software. Educated and qualified Certified Secure Software Lifecycle Professionals not only know how to implement security by writing secure code, but also how to meet security requirements, and design, test, and deploy secure software. In addition to understanding security concepts, they know how to balance security with risk, which is what software assurance is all about.

About (ISC)$^{2®}$

About the Author

Mano Paul, CSSLP, CISSP, AMBCI, MCAD, MCSD, Network+, ECSA is CEO and President of Express Certifications and SecuRisk Solutions, companies specializing in professional training, certification, security products and security consulting. His security experience includes designing and developing software security programs from Compliance-to-Coding, application security risk management, security strategy and management, and conducting security awareness sessions, training, and other educational activities. He is currently authoring the Official (ISC)^2 Guide to the CSSLP, is a contributing author for the Information Security Management Handbook, writes periodically for Certification, Software Development and Security magazines and has contributed to several security topics for the Microsoft Solutions Developer Network. He has been featured in various domestic and international security conferences and is an invited speaker and panelist in the CSI (Computer Security Institute), Catalyst (Burton Group), TRISC (Texas Regional Infrastructure Security Conference), SC World Congress, and the OWASP (Open Web Application Security Project) application security conferences. He can be reached at **mano.paul@expresscertifications.com** or **mano.paul@securisksolutions.com.**

---

a Dr. Dobb's Portal – Something's Gotta Give. http://www.ddj.com/architect/184414962

b Software Without Seat Belts – Certification Magazine. June 2008. http://www.certmag.com/read.php?in=3507

c Application Risk Modeling @ CSI 2007. http://securitymasala.wordpress.com/2007/11/26/application-risk-modeling-csi-2007/

d NIST 800-64 REV 1. Security Considerations in the Information Systems Development Life Cycle. http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf

e Howard, M. and Lipner, S., Security Development Lifecycle.