Chapter 2
Assimilating and Optimizing Software Assurance in the SDLC:
A Framework and Step-Wise Approach

Aderemi O. Adeniji
University of North Carolina at Charlotte, USA

Seok-Won Lee
University of North Carolina at Charlotte, USA

ABSTRACT

Software Assurance is the planned and systematic set of activities that ensures software processes and products conform to requirements while standards and procedures in a manner that builds trusted systems and secure software. While absolute security may not yet be possible, procedures and practices exist to promote assurance in the software lifecycle. In this paper, the authors present a framework and step-wise approach towards achieving and optimizing assurance by infusing security knowledge, techniques, and methodologies into each phase of the Software Development Lifecycle (SDLC).

INTRODUCTION

Software Assurance is steadily gaining ground in the Information Technology industry. The notion of proving secure software while supporting organization and system priorities is appealing to developers and customers alike. Software assurance aims to provide *justifiable confidence* that software is trusted to behave as intended even amidst intentional and unintentional attacks (Goertzel et al., 2007; Sinclair, 2005).

Based on experiences and lessons learned from designing a graduate level software assurance curriculum, assurance optimization is aided by implementing techniques in each phase of the SDLC. The intent of this paper is to share a strategy for integrating software assurance throughout the lifecycle in a methodical manner, proving a secure

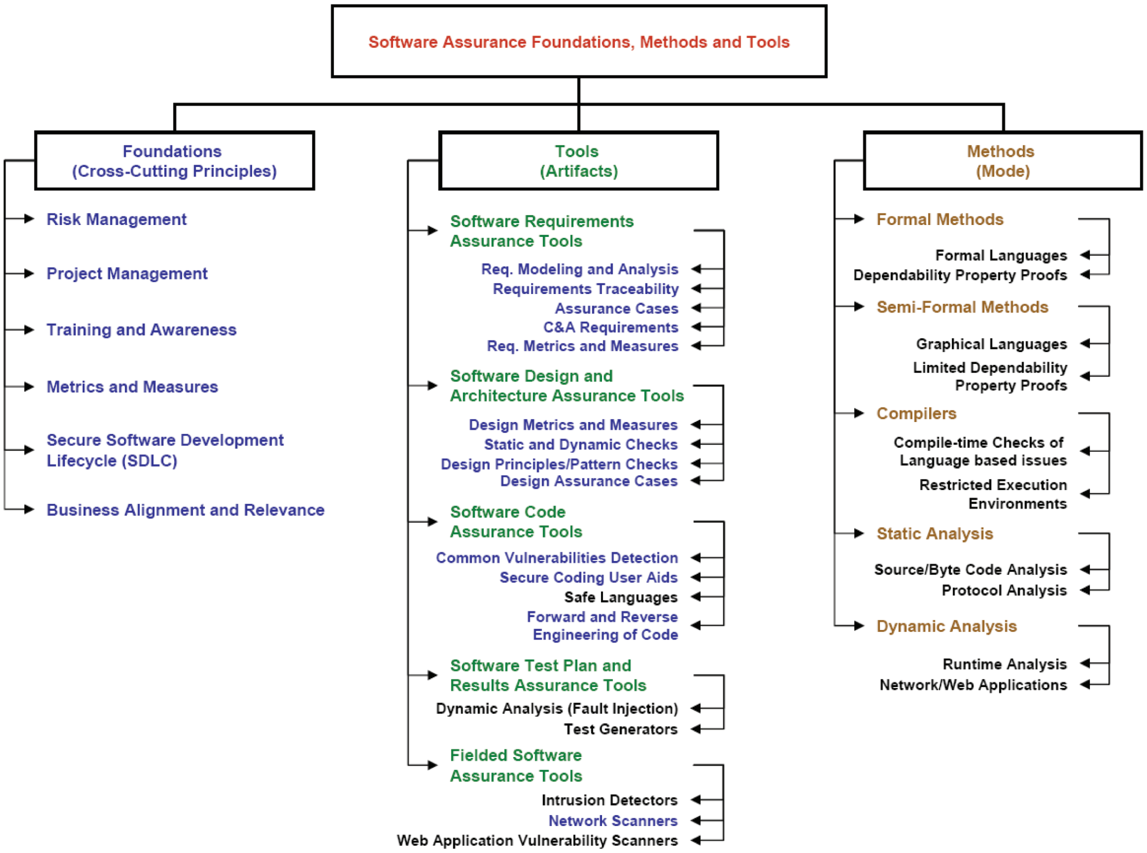
DOI: 10.4018/978-1-4666-1580-9.ch002

and trusted system. Several of the foundations, tools and methods used for optimization, shown on Figure 1, will be highlighted throughout the context.

BACKGROUND

Software is the core component of modern products and services, supporting business operations for all sectors of life. With each software use, there are factors which contribute to increased mission risk including: project size and complexity, attack sophistication, and use of third-party vendors (Ellison, 2006; McGraw, 2005). Dependence on this software makes security a primary concern (Allen et al., 2010). Software Assurance is achieved by understanding the mechanics of software built and/or acquired and incorporating validation tools and strategies into each phase of its lifecycle to build a trusted and secure product. Figure 2 diagrams this process, showing a step-wise approach for infusing assurance techniques into the SDLC by outlining approaches and artifacts produced. Knowledge gained from performing each step in a methodical and well-defined manner is carried forward, resulting in progressive learning. This is an iterative process, as education acquired from one phase will allow for more intelligent review in another. Assurance optimization can be achieved by mitigating common weaknesses in software throughout the aforementioned process. Peter G. Neumann identified nine sources of problems in computer systems (1994). A framework for

Figure 1. Software assurance foundations, methods and tools


Software Assurance Foundations, Methods and Tools
Foundations (Cross-Cutting Principles)
Risk Management
Project Management
Training and Awareness
Metrics and Measures
Secure Software Development Lifecycle (SDLC)
Business Alignment and Relevance
Tools (Artifacts)
Software Requirements Assurance Tools
Req. Modeling and Analysis
Requirements Traceability
Assurance Cases
C&A Requirements
Req. Metrics and Measures
Software Design and Architecture Assurance Tools
Design Metrics and Measures
Static and Dynamic Checks
Design Principles/Pattern Checks
Design Assurance Cases
Software Code Assurance Tools
Common Vulnerabilities Detection
Secure Coding User Aids
Safe Languages
Forward and Reverse Engineering of Code
Software Test Plan and Results Assurance Tools
Dynamic Analysis (Fault Injection)
Test Generators
Fielded Software Assurance Tools
Intrusion Detectors
Network Scanners
Web Application Vulnerability Scanners
Methods (Mode)
Formal Methods
Formal Languages
Dependability Property Proofs
Semi-Formal Methods
Graphical Languages
Limited Dependability Property Proofs
Compilers
Compile-time Checks of Language based issues
Restricted Execution Environments
Static Analysis
Source/Byte Code Analysis
Protocol Analysis
Dynamic Analysis
Runtime Analysis
Network/Web Applications