Carnegie Mellon

Software Security

Risk Management

Learning goals

- Review different security-focused software development processes
  - SDL processes
  - Process assessment
  - **Risk management**

- **Core argument**: Security must be addressed at every step of software development from inception to transition
  - Otherwise weaknesses will emerge

Recall: Managing security during development

- Adding security to a completed system is generally a poor approach
  - Security is (often) a systemic property
    - What are the security requirements for the system? What are the security threats? How will the system respond to identified threats?
- Better idea: design systems capable of resisting the threats they will face
  - Requires attention in all phases of software development
    - Requirements
    - Design
    - Implementation
    - Testing & Verification

Software processes and security management processes

- Ceremonies, artifacts, activities for quality work in a project
  - Scheduling activities
  - Sequencing activities
  - Foster planning and predictability
- Processes tend to evolve to towards that organization's needs
  - Internalize the strategy

Defining quality and quality goals

- Often framed in terms of generalities
  - “Implement all must have requirements”
  - “Satisfy the customer”
  - “Pass security assessment”
  - “Achieve all milestones”
- Or in terms of metrics
  - “Achieve x% statement coverage”
  - “Close all open defects”
- Progress is not indicative of quality!

Quality process

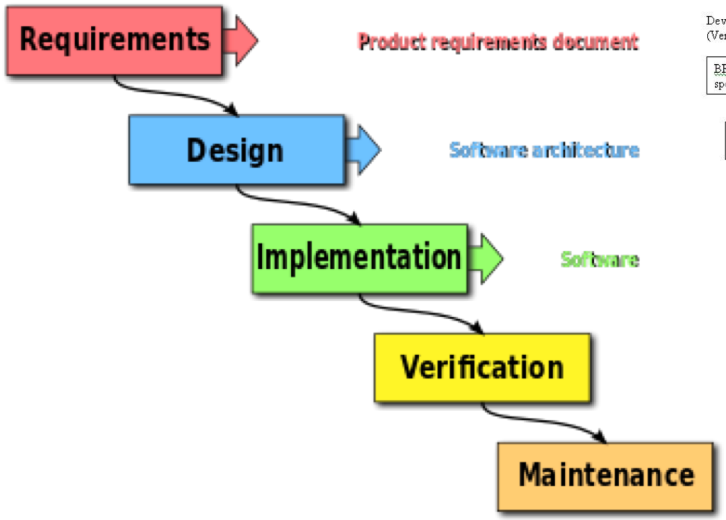
- Inseparable from software development process
- Strategies for achieving quality in a project

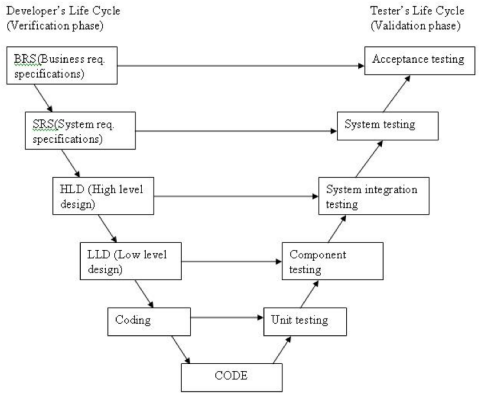
- People are an integral part of the process
  - Roles & responsibilities must be clear

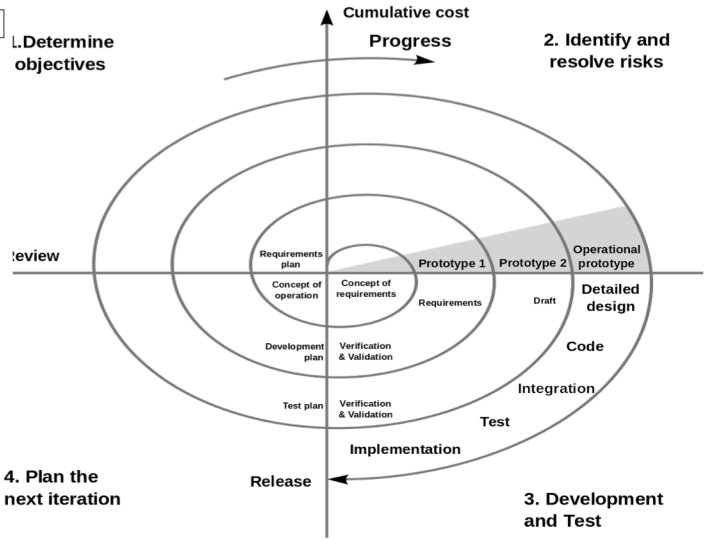
Quality process properties

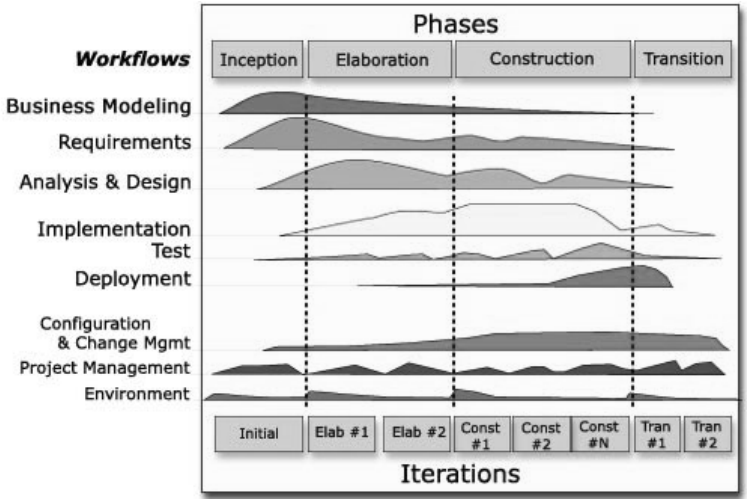
- **Completeness**: Activities designed to find important types of problems
- **Cost-conscious**: Working in the context of a project
- **Timely**: Problems found close to source
- **Visibility**: Are we making progress? Is the software ready to ship?
- **Measurable**: Can we tangibly improve

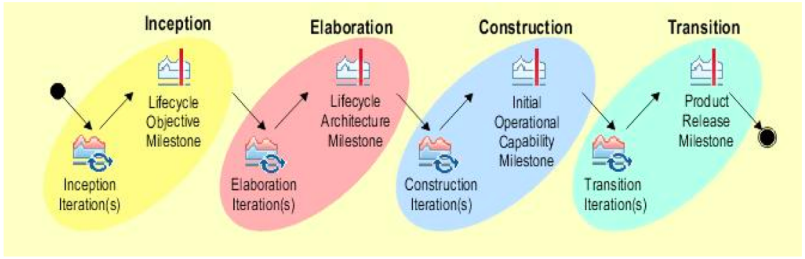
SDLC differences











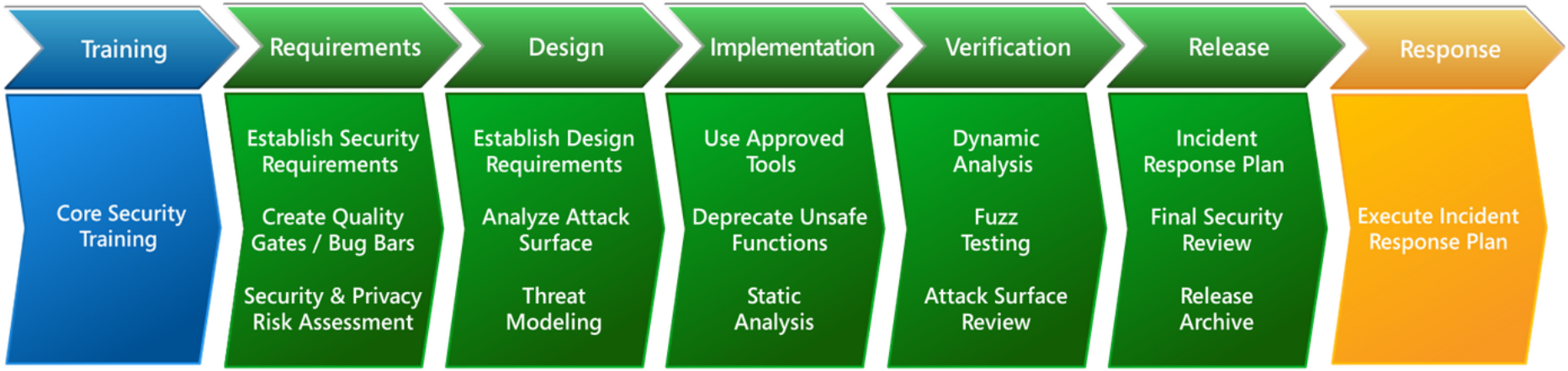
Choose your weapon wisely (J. Rockwood)

- Team Size
- Product Size and Complexity
- Total Developers
- Competent and Experienced Developers
- Level of Hacker Sentiment
- Management Style
- Organization-Wide Processes
- New Process Adoption
- Type of Product
- Requirements Stability
- Requirements Traceability

Choose your weapon wisely (J. Rockwood)

- Team Size
- Product Size and Complexity
- Total Developers
- Competent and Experienced Developers
- Level of Hacker Sentiment
- Management Style
- Organization-Wide Processes
- New Process Adoption
- Type of Product
- Requirements Stability
- Requirements Traceability

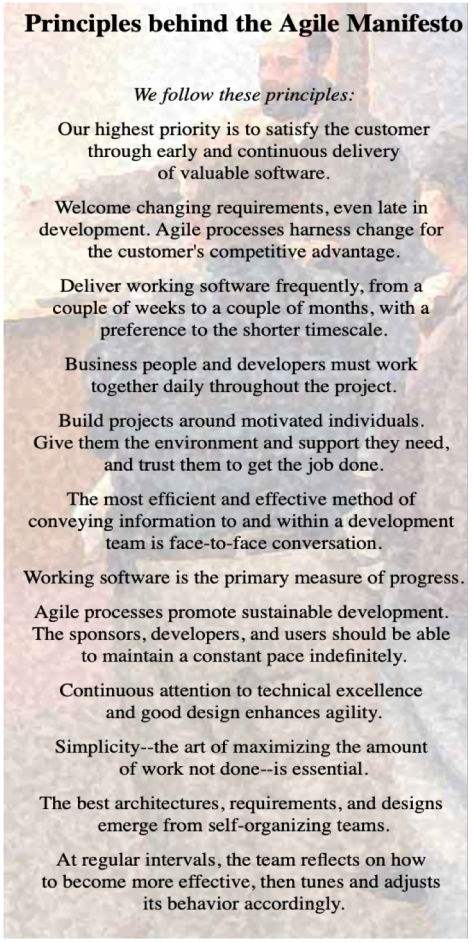
Microsoft's Security Development Lifecycle (SDL) process



http://www.microsoft.com/security/sdl

Agile processes

- Collection of lightweight processes
- SCRUM
- Extreme programming (XP) Crystal
- Feature-driven development (FDD)
- Dynamic system development method (DSDM)
- Kanban
- Lean software development
- Rapid application development (RAD)



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile and security

- Agile processes seem less managed than traditional software development lifecycles, but the reality is that the same amount of effort is simply refocused
  - Upon closer examination, there are ample opportunities to integrate software security practices
    - Secure Scrum

Compatibility of common security assurance activities with agile methods

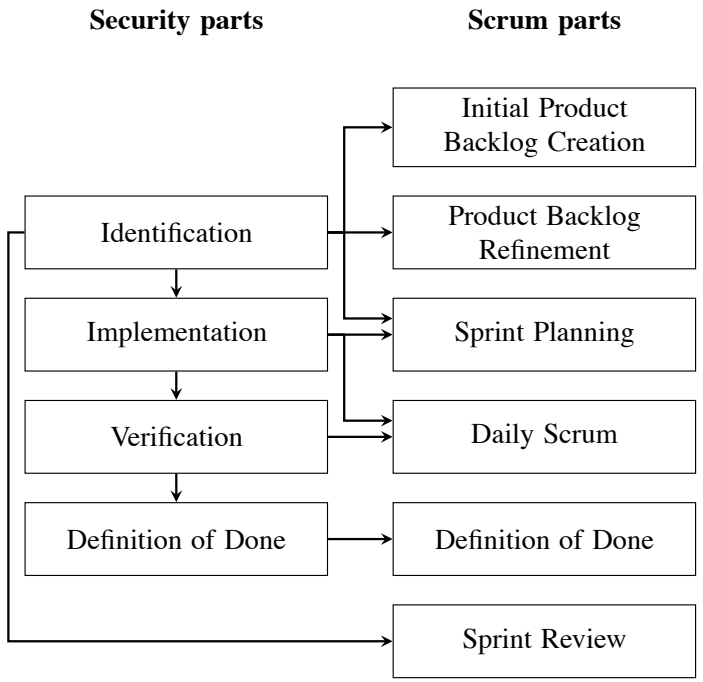
Security assurance method or technique		Match (2)	Indepen-dent (8)	(semi)-automated (4)	Mis-match (12)
Re-quire-ments	Guidelines		X		
	Specification analysis				X
	Review				X
Design	Application of specific architectural approaches		X		
	Use of secure design principles		X		
	Formal validation				X
	Informal validation				X
	Internal review	X			
	External review				X
Implementation	Informal correspondence analysis				X
	Requirements testing			X	
	Informal validation				X
	Formal validation				X
	Security testing			X	
	Vulnerability and penetration testing			X	
	Test depth analysis				X
	Security static analysis			X	
	High-level programming languages and tools		X		
	Adherence to implementation standards		X		
	Use of version control and change tracking		X		
	Change authorization				X
	Integration procedures		X		
	Use of product generation tools		X		
	Internal review	X			
	External review				X
	Security evaluation				X

Scrum

- *Scrum is a framework which people can use to address complex and adaptive problems while productively and creatively delivering products of the highest possible value*
  - Schwaber & Sutherland, 2017
- Roles
  - Product Owner: Business representative
  - Scrum Master: Manages the team & process
  - Development Team: Does everything else ;)

Secure SCRUM

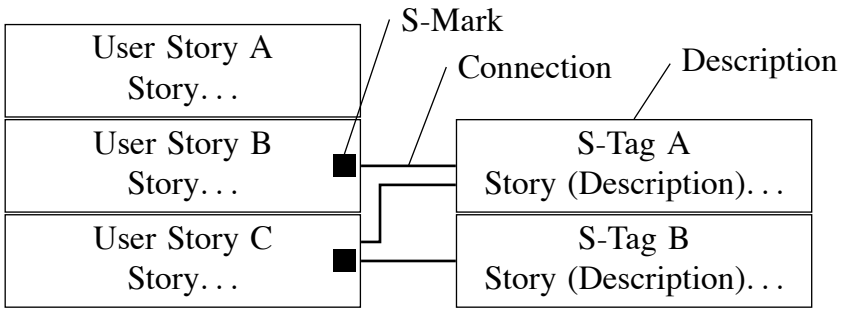
- Secure SCRUM integrates security activities into SCRUM
  - Identification
    - Mark security-relevant user stories
  - Implementation
    - Ensure developers are aware of security concerns
  - Verification
    - Verify security requirement implemented
  - Definition of Done
    - Established (security) criteria to declare product backlog item complete



Pohl, C., Hof, H.J. Secure Scrum: Development of Secure Software with Scrum. 2015

S-Tags and S-Marks

- Develop a common understanding of security risks in the Product Backlog
- Mark the story with an S-Mark that links to an S-Tag
  - An *S-Tag* describes a security concern
    - Security-related problem
    - Attack vector
    - Security principle that should be considered



Example: Attach an S-Tag to an S-Marked user story labelled “XSS”

- Indicate potential Cross Site Scripting attack so developers can mitigate

"*Definition of Done*" should include security

- *When a Product Backlog item or an Increment is described as "Done", everyone must understand what "Done" means. Although this may vary significantly per Scrum Team, members must have a shared understanding of what it means for work to be complete, to ensure transparency. This is the definition of "Done" for the Scrum Team and is used to assess when work is complete on the product Increment."*
  - "Definition of Done". In the Scrum Guide, the official body of knowledge of Scrum,

Definition of done with security

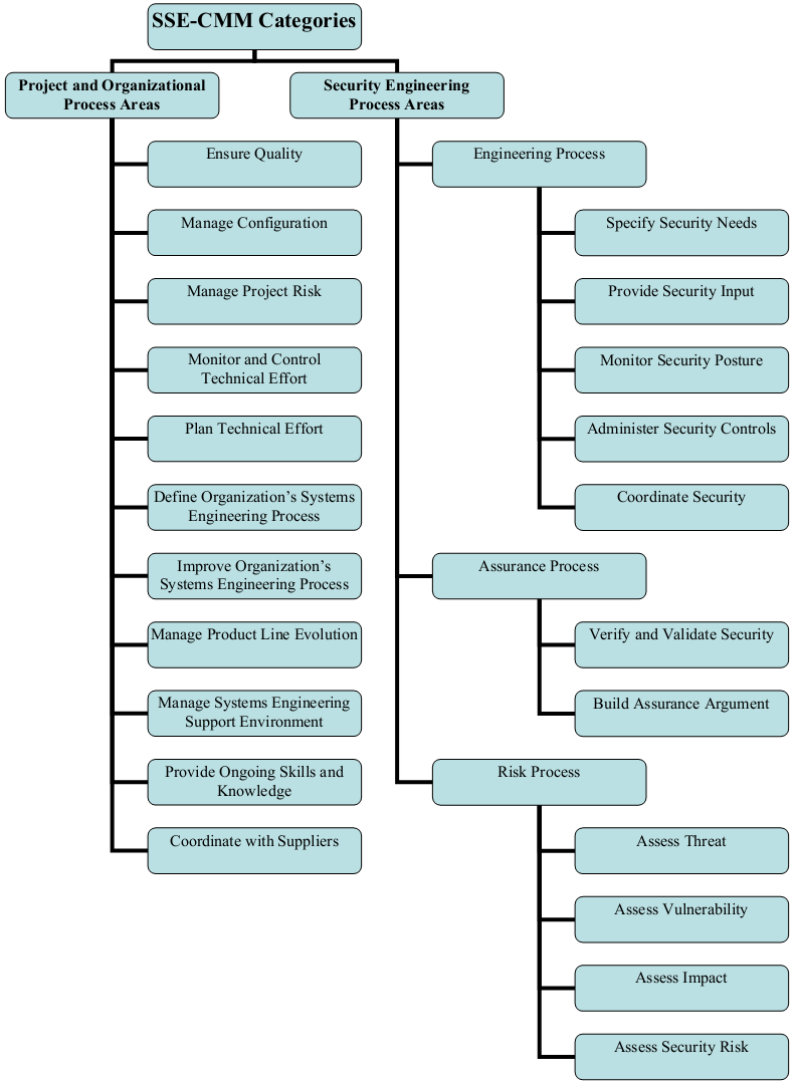
- Typical components
  - All unit tests have and passed
  - Deployment scripts have been written
  - Product backlog item assumptions have been fully satisfied
  - Code has been reviewed and approved
- Security components
  - Completed fuzzing (i.e. security tests)
  - Security-focused analysis tools completed
  - Security review completed

Assessment

- Mature organizations continually assess and improve their processes
  - Traditionally with regard to software development capabilities
    - See CMMI, ITIL, Six Sigma
- Security engineering capabilities should also be assessed

SSE-CMM

- Systems Security Engineering Capability Maturity Model
  - Assess the security engineering capability of an organization



http://www.sse-cmm.org

SSE-CMM Organizational Process Areas

- Define Organization's Security Engineering Process
- Improve Organization's Security Engineering Process
- Manage Security Product Line Evolution
- Manage Security Engineering Support Environment
- Provide Ongoing Skills and Knowledge
- Coordinate with Suppliers

SSE-CMM Project Process Areas

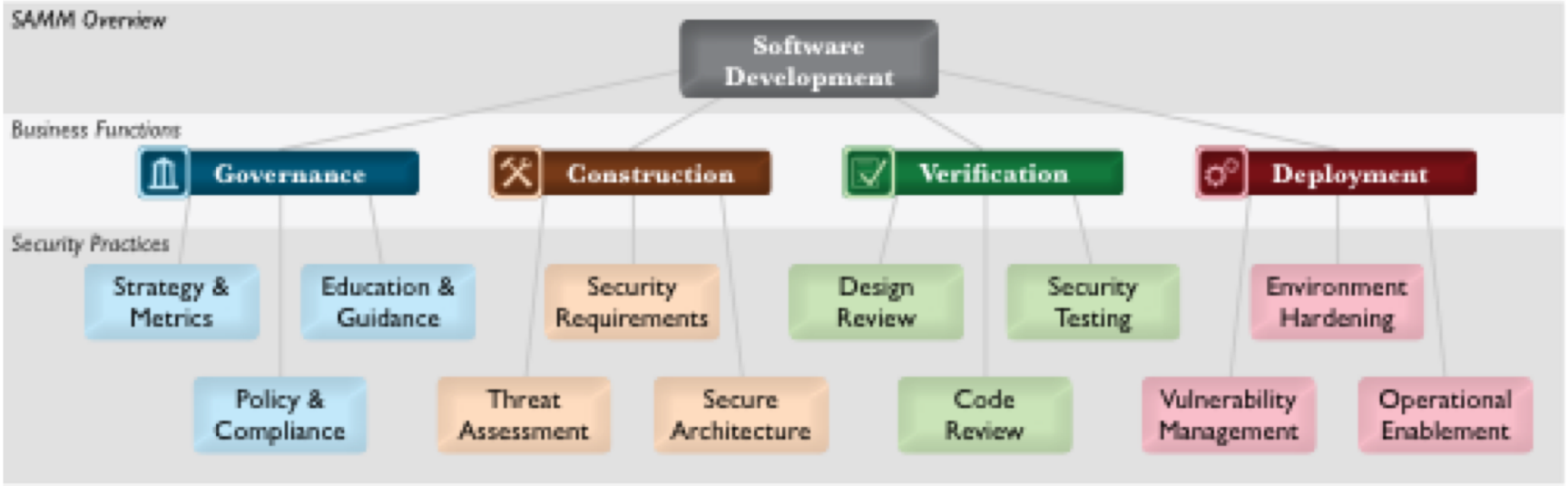
- Ensure Quality
- Manage Configurations
- Manage Program Risk
- Monitor and Control Technical Effort
- Plan Technical Effort

SSE-CMM Engineering Process Areas

- Administer Security Controls
- Assess Impact
- Assess Security Risk
- Assess Threat
- Assess Vulnerability
- Build Assurance Argument
- Coordinate Security
- Monitor Security Posture
- Provide Security Input
- Specify Security Needs
- Verify and Validate Security

Software Assurance Maturity Model (SAMM)

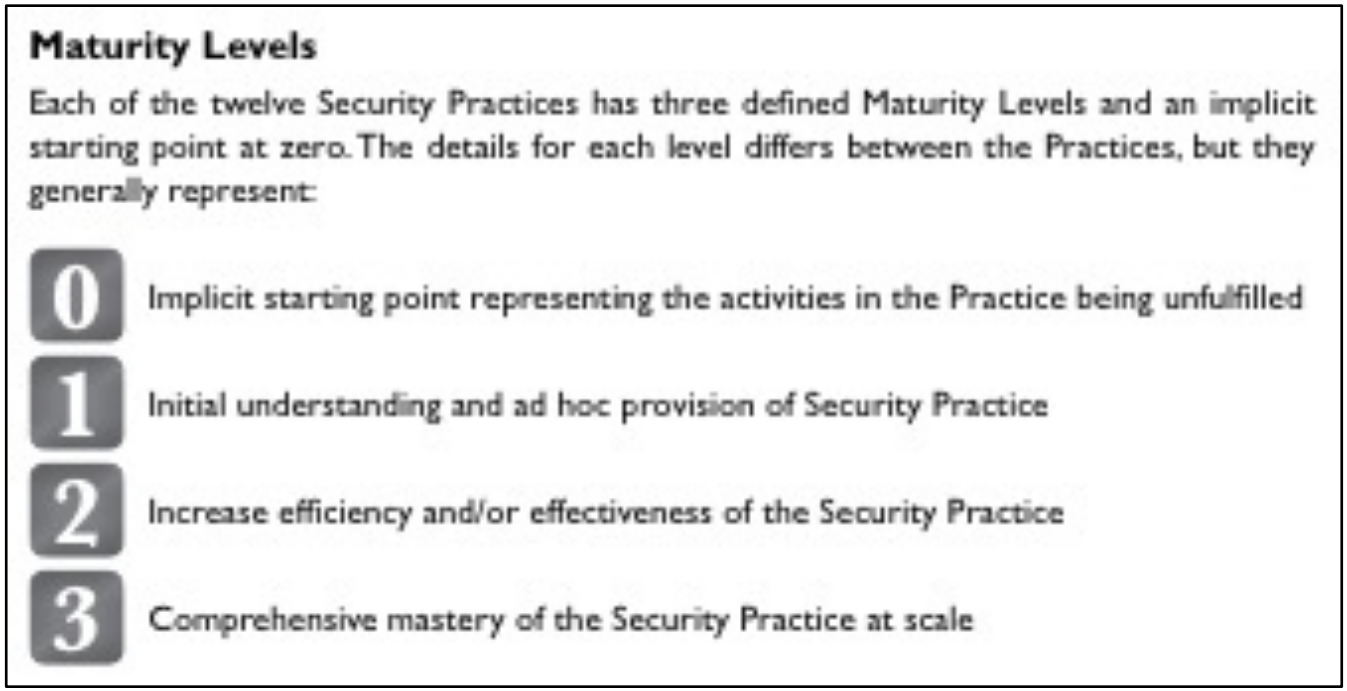
- Open model to evaluate software security practices
  - Build, improve, and measure software assurance activities



http://www.opensamm.org

Software Assurance Maturity Model (SAMM)

- Like other maturity models, SAMM assigns levels to capabilities



Maturity Levels

Each of the twelve Security Practices has three defined Maturity Levels and an implicit starting point at zero. The details for each level differs between the Practices, but they generally represent:

0 Implicit starting point representing the activities in the Practice being unfulfilled

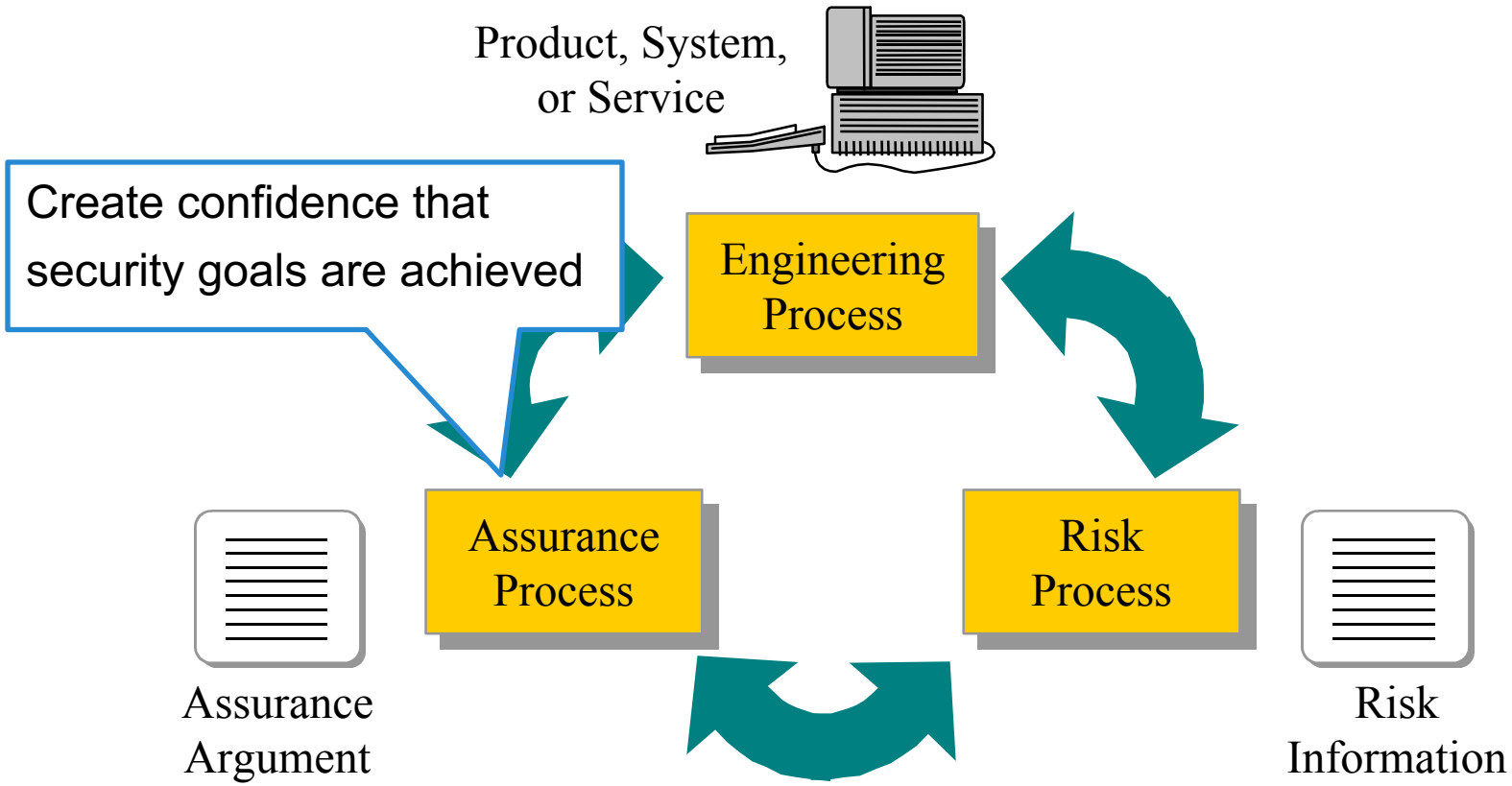
1 Initial understanding and ad hoc provision of Security Practice

2 Increase efficiency and/or effectiveness of the Security Practice

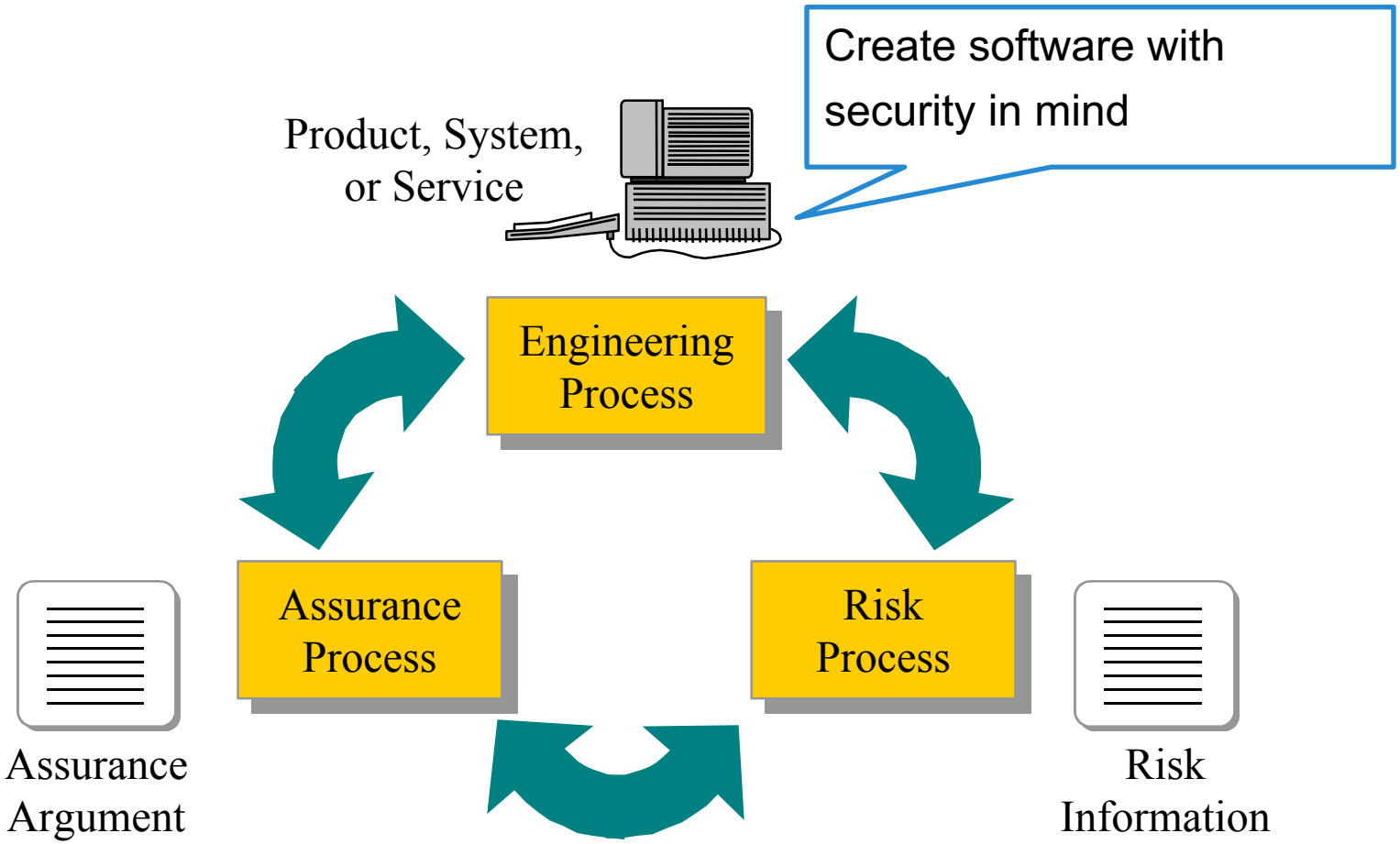
3 Comprehensive mastery of the Security Practice at scale

http://www.opensamm.org

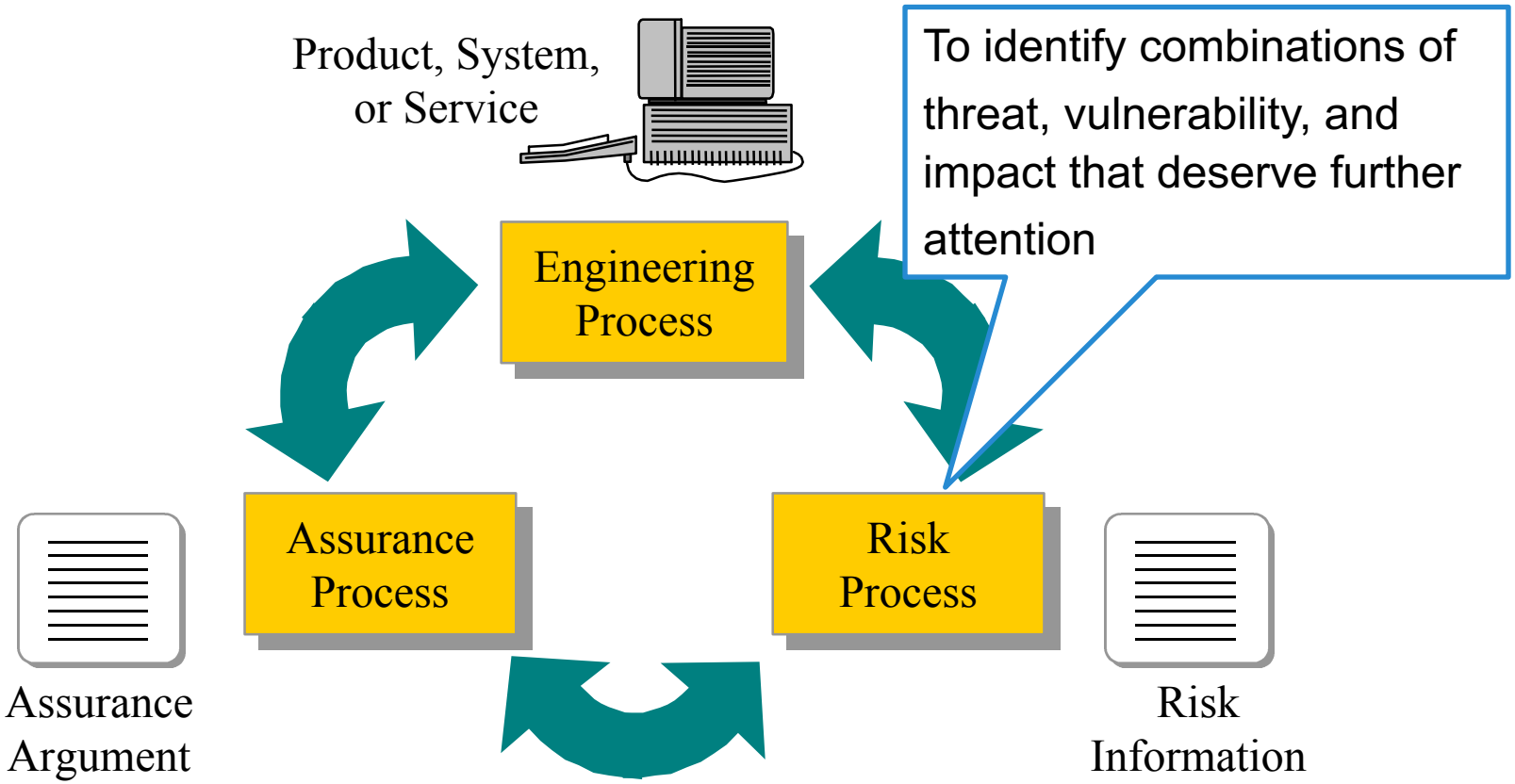
Security Engineering Process



Security Engineering Process



Security Engineering Process



Software Security Framework (SSF)

- Built on SAMM
- 12 practices organized into four domains

Governance	Intelligence	SDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

https://www.informit.com/articles/article.aspx?p=1271382

SSF Governance

Practices that help organize, manage, and measure a software security initiative.

- *Strategy and metrics:* encompasses planning, assigning roles and responsibilities, identifying software security goals, determining budgets, and identifying metrics and gates
- *Compliance and policy* practice is focused on identifying controls for compliance
  - Regulatory/contract obligations
- *Training* for developers and architects

SSF Intelligence

- Practices that gather corporate security knowledge
  - *Attack models* capture information used to think like an attacker
  - *security features and design* practice is charged with creating usable security patterns for major security controls
  - The *standards and requirements* practice involves eliciting explicit security requirements from the organization

SSF SDL Touchpoints

- Essential software security best practices that are integrated into the SDLC
  - *Architecture analysis* encompasses capturing software architecture in concise diagrams, applying lists of risks and threats, adopting a process for review, and building an assessment and remediation plan for the organization
  - *Code review* practice integrates security into code review tools, development of customized rules, profiles for tool use by different role
  - *Security testing* integrates security into standard quality assurance processes

SSF Deployment

- Practices that interface with traditional network security and software maintenance organizations
  - *Penetration testing* involves more standard system-level testing carried out by security specialists
  - *Software environment* practice concerns itself with the underlying system security (OS, network, etc.)
  - *Configuration management and vulnerability management* practice application patching and updating applications, version control, defect tracking and remediation, and incident handling

Using SSF

- Judge organizational maturity based on practices
  - Assessment methodology rather than prescriptive process

Correctness by Construction

- "When you buy a car, you expect it to work properly. You expect the manufacturer to build the car so that it's safe to use, travels at the advertised speed, and can be controlled by anyone with normal driving experience. When you buy a piece of software, you would like to have the same expectation that it will behave as advertised. Unfortunately, conventional software construction methods do not provide this sort of confidence …"
  - Anthony Hall and Roderick Chapman, Praxis Critical Systems

Hall, Anthony & Chapman, Roderick. "Correctness by Construction: Developing a Commercial Secure System." *IEEE Software 19*, 1 (Jan./Feb. 2002): 18-25.

Correctness by Construction

- Each development phase/step is rigorously defined with clear goals and a focus on removing defects as early as possible
  - Heavy reliance on formal methods
    - Rigorous formality is the only way to achieve the necessary precision
      - Notably, the Z formal specification language

Seven principles

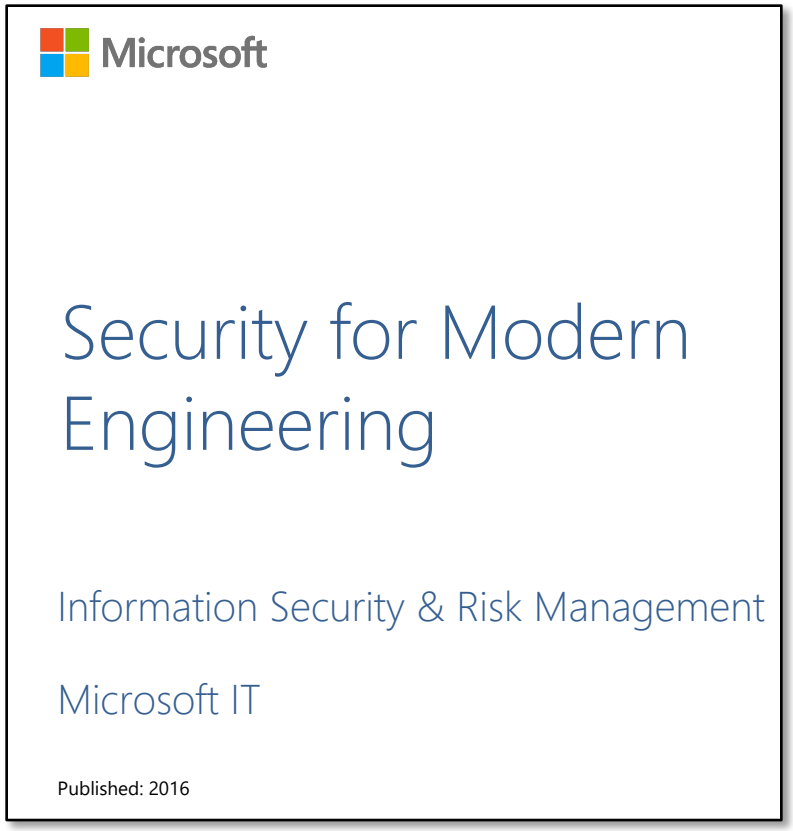
1. Expect requirements to change
   - Changing requirements are managed by adopting an incremental approach and paying increased attention to design to accommodate change. Apply more rigor, rather than less, to avoid costly and unnecessary rework.
2. Know why you're testing
   - Recognize that there are two distinct forms of testing, one to build correct software (debugging) and another to show that the software built is correct (verification). These two forms of testing require two very different approaches.
3. Eliminate errors before testing
   - Better yet, deploy techniques that make it difficult to introduce errors in the first place. Testing is the second most expensive way of finding errors. The most expensive is to let your customers find them for you.

Seven principles

4. Write software that is easy to verify
5. Develop incrementally
   - Making small changes makes the software much easier to verify
6. Some aspects of software development are just plain hard
   - There is no silver bullet
7. Software is not useful by itself
   - The executable software is only part of the picture. It is of no use without user manuals, business processes, design documentation, well-commented source code, and test cases

Microsoft's Modern Engineering

- Founded on SDL but more focused on general processes

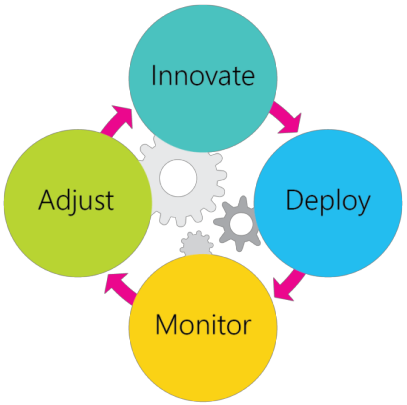


Microsoft: Security for Modern Engineering. 2016

Microsoft approach

- Embrace DevOps culture
  - Speed up the pace of innovation by shortening the release cycles using agile methodology
  - Enabling faster feedback loops from customers which can result in application features and bug fixes.

- Security is traditionally riddled with checkpoints and hard *gates*
  - Slows down process



Microsoft: Security for Modern Engineering. 2016

Continuous assurance

- Embed security processes in other development processes
  - Make security inseparable from development
  - Leverage automation
  - Everyone becomes a security expert
- Technical Control Procedures
  - Required security controls

Auditing and logging	Critical systems may require both features and process to be implemented inorder to efficiently and effectively monitor the security health of the application ecosystem. This would enable you to identify and respond to inappropriate activity within the application ecosystem.
Authentication	"Who are you?" Authentication is the process where an entity proves the identity of another entity, typically through credentials, such as a user name and password. The controls in this category provides both technical and design considerations ne essary to implement robust authentication
Authorization	"What can you do?" Authorization is how application provides access controls for resources and operations. The controls in this category represent both Microsoft authorization requirements as well as specific technical and operational details that have been found to be critical for effective authorization for the application.
Application business logic	Every application has a different business process, application specific logic and can be manipulated in an infinite number of combinations. Think about abused and misuse cases in your application and how you can protect against them.
Configuration setting	There is a delicate balance between default configuration settings; configuration settings required for development; and those that are appropriate in production. Misconfigura-tion can lead directly to an exploit, but more often misconfiguration provides information disclosure and elevation of privileges that would otherwise not be easily accomplished. System complexity contributes with stings that may not always be secure by default.
Cryptography	Cryptographic methods are utilized to provide protection against offline or third party attacks in at-rest or in-transit scenarios. Choosing the correct cryptographic methods provides secrecy; proof of integrity, and supplies complex transformations that provide a strong basis for trustworthy in protocols for in-transit and at-rest data.
Data handling	All sensitive data and secrets must be protected when in transit and when at rest. Secrets such as credentials, keys, passwords, and other secrets cannot be stored in plain text (especially in configuration files). Unencrypted, these secrets are prone to discovery and can easily allow a partially compromised application or system to be further exploited, thus increasing the impact of the exploit.
Input validation	Constrain, reject, and sanitize user controlled and user controllable data before being consumed by the application ecosystem.
Output encoding	HTML, XML, LDAP, etc. allows text to be rendered within executable context, not just as static text. Output Encoding ensures that user controlled data is rendered as static text by removing special characters that would otherwise allow for arbitrary command execution.

Microsoft: Security for Modern Engineering. 2016

Risk management

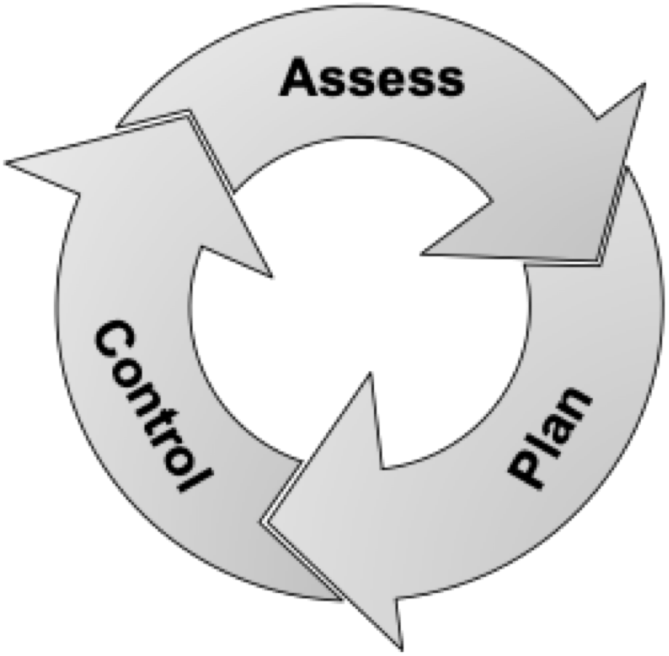
- Risk is the possibility of suffering loss
- Well run software projects actively manage risk by identifying and mitigating them
- Software security risks include
  - Risks found in the outputs and results produced by each life-cycle phase during assurance activities
  - Risks introduced by insufficient processes
  - Personnel-related risks

Risk tolerance and security

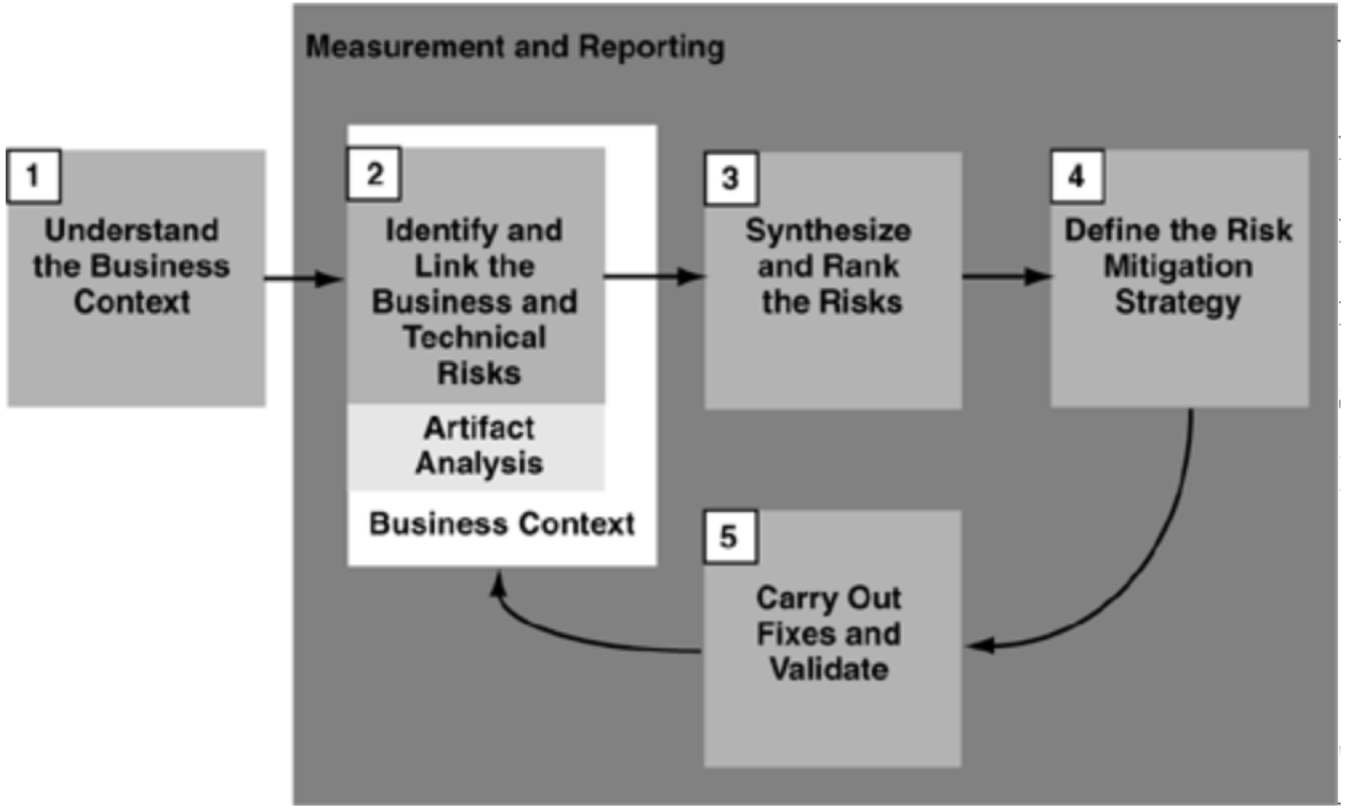
- Simple question: How much security is enough?
  - What are the critical assets and business processes that support achieving our organizational goals?
  - Under which conditions and with what likelihood are assets and processes at risk?
  - When risks go beyond these thresholds, which mitigating actions do we need to take and with which priority?
  - For unacceptable, which protection strategies do we need to put in place?
  - How well are we managing our security state today?

Risk process

- Assess risk
  - Transform the concerns people have into distinct, tangible risks that are explicitly documented and analyzed
- Plan for risk control
  - Determine an approach for addressing each risk; produce a plan for implementing the approach
- Control risk
  - Deal with each risk by implementing its defined control plan and tracking the plan to completion



Risk management framework for security



Step 1. Understand the business context

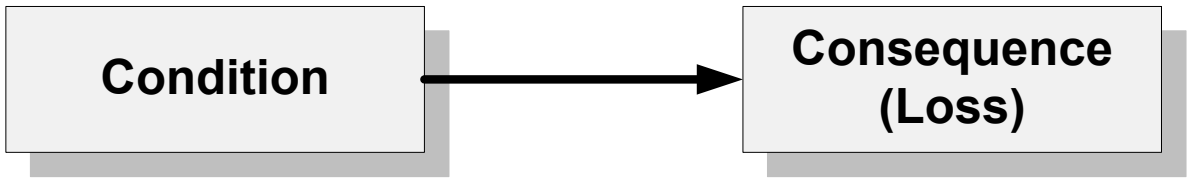
- Extract and describe business goals, priorities, and circumstances to understand which kinds of software risks are important to care about and which business goals are paramount
  - Without a clear and compelling tie to either business or mission consequences, technical risks, software defects, and the like are not often compelling enough on their own to warrant action.
  - Unless software risks are described in terms that business people and decision makers understand, they will likely not be addressed.
- Example: maintain service availability

Step 2. Identify business and technical risks

- Business risks: Threat to business goals
  - Financial loss
  - Damage to reputation
- Technical risks: Threat to system implementation
  - Unexpected or undefined behavior
  - Reliability issues and system crashes
  - Missing controls

Risk format

- A condition that directly produces a loss or adverse consequence.
  - No uncertainty exists.
  - The condition exists and is having a negative effect on performance.
- Issues can also lead to (or contribute to) other risks by
  - Creating a circumstance that enables an event to trigger additional loss
  - Making an existing event more likely to occur
  - Aggravating the consequences of existing risks



Risk Register

- Well-formatted risks stated in condition; consequence form
  - Condition and consequence are both statements of fact
  - Mitigation activities should be actionable

Business Risk

Risk ID	Condition	Consequence	Prob	Imp	Rating	Mitigation Activities
R-01	Communication with remote employees is difficult	Critical software elements not completed on time	2	5	10	Establish weekly status conference calls with remote team
R-02	Hardware damaged during development	Damaged hardware causes inaccurate results	1	5	5	Dedicate safety engineer to monitor and repair hardware

Technical Risk

Step 3. Prioritize risks

- Probability * Impact = Rating

Risk ID	Condition	Consequence	Prob	Imp	Rating	Mitigation Activities
R-01	Communication with remote employees is difficult	Critical software elements not completed on time	2	5	10	Establish weekly status conference calls with remote team
R-02	Hardware damaged during development	Damaged hardware causes inaccurate results	1	5	5	Dedicate safety engineer to monitor and repair hardware

Step 4. Mitigation

- Strategy for mitigating the highest-priority risks in a cost-effective manner

Risk ID	Condition	Consequence	Prob	Imp	Rating	Mitigation Activities
R-01	Communication with remote employees is difficult	Critical software elements not completed on time	2	5	10	Establish weekly status conference calls with remote team
R-02	Hardware damaged during development	Damaged hardware causes inaccurate results	1	5	5	Dedicate safety engineer to monitor and repair hardware

Step 5. Validation

- Execute mitigation strategies
- Risk management is a continuous activity

Summary

- Software security must be part of the project process
- Planned and monitored from inception to transition
- Different process methodologies approach security in accordance with defined practices
  - Agile emphasizes communication and coordination
  - Traditional methods emphasize security in each phase