Software Security and Quality Assurance (SSQA) Framework Guidance

Understanding the SSQA Framework
This guidance document is owned by Ministry of Transport and Communications (MOTC) who shall update as necessary.
Table of Contents

Introduction ................................................................................................................................. 6
Software Security and Quality Assurance Framework ......................................................... 8
Aligning the SSQA with the National Information Assurance Framework (NIAF) .................. 9
Understanding the Framework ............................................................................................... 10
The Software Security Framework ......................................................................................... 11
  Governance ............................................................................................................................. 11
  Intelligence ............................................................................................................................. 11
SSDL Touchpoints .................................................................................................................. 12
Deployment ............................................................................................................................... 12
Objectives, Activities & Metrics .............................................................................................. 13
Maturity Ratings ....................................................................................................................... 13
Reading the Objectives and Activities Table ......................................................................... 14
Governance .............................................................................................................................. 15
Strategy and Metrics ............................................................................................................... 15
Compliance and Policy ........................................................................................................... 16
Training .................................................................................................................................... 17
Intelligence ............................................................................................................................... 18
  Attack Models ....................................................................................................................... 18
  Security Features and Design ............................................................................................... 19
  Standards and Requirements ................................................................................................. 20
SSDL Touchpoints .................................................................................................................. 21
  Architecture Analysis .......................................................................................................... 21
  Code Review ......................................................................................................................... 22
  Security Testing ................................................................................................................... 23
Deployment ............................................................................................................................... 24
  Penetration Testing .............................................................................................................. 24
  Software Environment ......................................................................................................... 25
  Configuration Management and Vulnerability Management ............................................. 26
What is a Software Security Group (SSG)? ........................................................................ 27
What is meant by a Satellite? ................................................................................................. 27
Roles and Responsibilities ....................................................................................................... 28
Appendix: SSQA Domains, Practices and Activities ............................................................... 31
Legal Mandate(s)

Emiri decision No. (8) for the year 2016 sets the mandate for the Ministry of Transport and Communication (hereinafter referred to as “MOTC”) provides that MOTC has the authority to supervise, regulate and develop the sectors of Information and Communications Technology (hereinafter “ICT”) in the State of Qatar in a manner consistent with the requirements of national development goals, with the objectives to create an environment suitable for fair competition, support the development and stimulate investment in these sectors; to secure and raise efficiency of information and technological infrastructure; to implement and supervise e-government programs; and to promote community awareness of the importance of ICT to improve individual’s life and community and build knowledge-based society and digital economy.

Article (22) of Emiri Decision No. 8 of 2016 stipulated the role of the Ministry in protecting the security of the National Critical Information Infrastructure by proposing and issuing policies and standards and ensuring compliance.

This guideline has been prepared taking into consideration current applicable laws of the State of Qatar. In the event that a conflict arises between this document and the laws of Qatar, the latter, shall take precedence. Any such term shall, to that extent be omitted from this Document, and the rest of the document shall stand without affecting the remaining provisions. Amendments in that case shall then be required to ensure compliance with the relevant applicable laws of the State of Qatar.
References

- [IAP-NAT-DCLS] National Information Classification Policy
- [IAP-NAT-IAFW] Information Assurance Framework
- [NIAF-SSQA-S-SSL1] – SSQA Level 3 Software Security Standard

A glossary of terms is defined within the Information Assurance Framework, [IAP-NAT-IAFW].
Introduction

An ever-increasing reliance upon digital services and technology coupled with the ongoing discovery of weaknesses, which threaten the confidentiality, integrity and availability of digital services and the data; presents an ongoing challenge for organizations and governments worldwide.

It is therefore important to make sure that the software applications used by these entities are as secure as possible, ensuring that security has been considered within development or procurement efforts and also ensuring that such concerns are considered as part of outsourced development efforts by third-parties.

Secure development is a practice to ensure that the code and processes that go into developing applications are as secure as possible.

Secure development practices consider security during each development phase or stage, regardless of chosen development methodology.

Within the context of software security, secure development considers the application of secure coding practices to transform the traditional Software Development Lifecycle (SDL) into a Secure Software Development Lifecycle (SSDL).

This guidance document describes the Software Security and Quality Assurance (SSQA) Framework, integrated within the National Information Assurance Framework (NIAF) and describes how the domains, practices and activities help to support the ideal of developing and sourcing secure software solutions to enable the State of Qatar to thrive as a digital nation.
Scope
This standard applies to all Agencies engaged in the development or implementation of software solutions, including those which outsource or procure software or digital services.

Purpose
This guidance document aims to introduce constituents to the Software Security and Quality Assurance (SSQA) Framework, outlining how the framework integrates with the State of Qatar’s National Information Assurance Framework (NIAF) and National Information Security Compliance Framework (NISCF).

While introducing the SSQA and explaining the frameworks placement alongside existing information assurance initiatives, this guidance document also aims to explore the core domains, the practices and the activities that the SSQA outlines to build and enhance the security of software development and procurement activities. These components are explained alongside key roles and responsibilities that will be necessary for the effective implementation of controls outlined within the Software Security Standards.
Software Security and Quality Assurance Framework

The Software Security and Quality Assurance (SSQA) Framework is based on the widely accepted standard BSIMM (Building Security in Maturity Model).

The purpose of the BSIMM is to quantify the activities carried out by real software security initiatives. Because these initiatives use different methodologies and different terminology, the BSIMM requires a framework that allows a description of all the initiatives in a uniform way. The Software Security Framework (SSF) and activity descriptions provide a common vocabulary for explaining the salient elements of a software security initiative, thereby allowing a comparison between initiatives that use different terms, operate at different scales, exist in different vertical markets, or create different work products.
Aligning the SSQA with the National Information Assurance Framework (NIAF)

The National Information Assurance Framework (NIAF) is the primary driver directing constituents of the State of Qatar towards the secure use of data and digital technologies, enabling a digital society. The National Information Assurance Policy (NIAP), a key instrument within the assurance framework outlines software security and the development or procurement of secure software to as a key security domain.

The NIAF provide a hierarchical model for the development and communication of information governing digital security.

The Software Security and Quality Assurance (SSQA) Framework provides standards and guidance to support the achievement of the software security direction provided by the NIAP. Software security controls are separated across the SSQA standards and should be considered similarly to the National Information Assurance (NIA) control set, which includes mandatory level 1 (baseline) controls and optional Level 2 and 3 (recommended) controls (which may be required dependent upon the criticality of specific assets).

The SSQA framework specifically supports the intent and direction provided by statements within the NIAP, however, the SSQA standards are also fundamental to the success of the National Information Security Compliance Framework (NISCF) which is charged with assuring the implementation and effectiveness of controls within the NIAF.
Understanding the Framework
At the center of the framework sits the SSA Guidance Document. This guidance document provides the objectives, activities to achieve these objectives, maturity scale for measuring improvement and success factors for the agencies/teams/projects to utilize.

The Guidance Structure
At the highest level, there exists four critical domains, Governance, Intelligence, SSDL Touchpoints and Deployment.

These domains house the practices and their respective activities that comprise the Software Security Framework (SSF).

For each domain, the framework defines three practices. Each practice is an area of activities that build assurance for the related business function.

There are twelve practices that are the independent silos for improvement that map to the four business functions of software development.

Within each practice, the framework defines three maturity levels as objectives. Each level is characterized by a successively more sophisticated objective defined by specific activities, and more stringent success metrics than the previous level.

Each practice can be improved independently, though related activities can lead to optimizations due to overlaps and inter dependencies.
The Software Security Framework

**Governance**

Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.

<table>
<thead>
<tr>
<th>Governance</th>
<th>Compliance &amp; Policy</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy &amp; Metrics encompases planning, assigning roles and responsibilities, identifying software security goals, determining budgets, and identifying metrics and gates.</td>
<td>Focuses on identifying controls for compliance regimens such as PCI DSS and HIPAA, developing contractual controls such as service level agreements to help control COTS software risk, setting organizational software security policy, and auditing against that policy.</td>
<td>Has always played a critical role in software security because software developers and architects often start with very little security knowledge.</td>
</tr>
</tbody>
</table>

**Intelligence**

Practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization. Collections include both proactive security guidance and organizational threat modelling.

<table>
<thead>
<tr>
<th>Intelligence</th>
<th>Security Features &amp; Design</th>
<th>Standards &amp; Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack Models capture information used to think like an attacker: threat modelling, abuse case development and refinement, data classification, and technology-specific attack patterns.</td>
<td>is charged with creating usable security patterns for major security controls (meeting the standards defined in the Standards &amp; Requirements practice), building middleware frameworks for those controls, and creating and publishing other proactive security guidance.</td>
<td>involves eliciting explicit security requirements from the organization, determining which COTS to recommend, building standards for major security controls (such as authentication, input validation, and so on), creating security standards for technologies in use, and creating a standards review board.</td>
</tr>
</tbody>
</table>
SSDL Touchpoints
Practices associated with analysis and assurance of particular software development artefacts and processes. All software security methodologies include these practices.

<table>
<thead>
<tr>
<th>SSDL Touchpoints</th>
<th>Code Review</th>
<th>Security Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture Analysis</strong></td>
<td>includes the use of code review tools, development of tailored rules, customized profiles for tool use by different roles (for example, developers versus auditors), manual analysis, and tracking/measuring results.</td>
<td>is concerned with pre-release testing, including integrating security into standard quality assurance processes. The practice includes use of black box security tools (including fuzz testing) as a smoke test in QA, risk-driven white box testing, application of the attack model, and code coverage analysis. Security testing focuses on vulnerabilities in construction.</td>
</tr>
<tr>
<td>encompasses capturing software architecture in concise diagrams, applying lists of risks and threats, adopting a process for review (such as STRIDE or Architecture Risk Analysis), and building an assessment and remediation plan for the organization.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Deployment
Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security.

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Software Environment concerns itself with OS and platform patching, web application firewalls, installation and configuration documentation, application monitoring, change management and, ultimately, code signing.</th>
<th>Configuration Management &amp; Vulnerability Management concerns itself with patching and updating applications, version control, defect tracking and remediation, and incident handling.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Penetration Testing</td>
<td>involves standard outside-in testing of the sort carried out by security specialists. Penetration testing focuses on vulnerabilities in the final configuration and provides direct feeds to defect management and mitigation.</td>
<td></td>
</tr>
<tr>
<td><strong>Software Environment</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>**Configuration Management &amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Vulnerability Management</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Objectives, Activities & Metrics
Each of the 12 Security Practices within the 4 Business Functions have three maturity levels (discussed below). Each level has several components that specify the critical factors for understanding and achieving the stated maturity level. Maturity ratings are used to benchmark current practices and draw out a plan for the future.

Objective
The objective is a general statement that captures the assurance goal of attaining the associated level. As the levels increase for a given practice, the objectives characterize more sophisticated goals in terms of building assurance for software development, deployment, and operations.

Activities
The activities are core requisites for attaining the level. Some are meant to be performed organization-wide, and some correspond to actions for individual project teams. In either case, the activities capture the core security function, and MOTC agencies are free to determine how they fulfil the activities.

Audit Points
The audit points provide a general guidance about the questions the agency/program should be asking to test whether the objectives are being fulfilled.

Maturity Ratings
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
Reading the Objectives and Activities Table

In the following section, the framework describes the objectives of each security practice and the activities associated with it.

The following image describes how to read this table:

<table>
<thead>
<tr>
<th>Maturity Level (increasing 1&gt;3)</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td>This section describes the objective of the corresponding maturity rating for the security practice</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Activities</strong></td>
<td>This section describes the high-level activities are needed to meet the required level of corresponding maturity</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Audit Considerations</strong></td>
<td>These are the questions you can ask to test whether the objective has been met or not</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Governance
#### Strategy and Metrics

The Strategy & Metrics practice encompasses planning, assigning roles and responsibilities, identifying software security goals, determining budgets, and identifying metrics and gates.

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>SM1</th>
<th>SM2</th>
<th>SM3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>To establish a software development process with appropriate checkpoints for assessing security and developing understanding amongst of software security risks and mitigation approaches at the executive level.</td>
<td>To develop internal interest in the security of software development, manage internal efforts through security metrics and formally accept security risks.</td>
<td>To track the progress of software development and maintain awareness of scheduled security activities.</td>
</tr>
<tr>
<td>Activities</td>
<td>Publish process (roles, responsibilities, plan), evolve as necessary; Create evangelism role and perform internal marketing; Educate executives; and, Identify gate locations, gather necessary artefacts.</td>
<td>Publish data about software security internally; Enforce gates with measurements and track exceptions; Create or grow a satellite; Identify metrics and use them to drive budgets; and, Require security sign-off.</td>
<td>Use an internal tracking application with portfolio view; and, Run an external marketing program.</td>
</tr>
<tr>
<td>Audit Considerations</td>
<td>Has the organization published a software development process? Have roles, responsibilities and activities been explicitly defined? Are executives routinely educated to understand software security risks? Are executives made aware of external approaches to attaining software security?</td>
<td>Are software security metrics to senior management? Are Security Gates enforced and are exceptions tracked? Is security risk formally accepted prior to release? Are individuals with an interest in software security connected?</td>
<td>Is software development tracked via a centralized application? Are metrics developed Software Security Group (SSG)? Does the SSG develop external resources to promote the SSDL?</td>
</tr>
</tbody>
</table>
## Compliance and Policy

The Compliance & Policy practice is focused on identifying controls for compliance regimens such as PCI-DSS and HIPAA, developing contractual controls such as service level agreements to help control COTS (Commercial-Off-The-Shelf) software risk, setting organizational software security policy, and auditing against that policy.

### Compliance and Policy (CP)

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>CP1</th>
<th>CP2</th>
<th>CP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>To document and understand legislative, regulatory and customer-driven security and privacy requirements and formalize a software security policy that provides a unified approach to satisfying such requirements.</td>
<td>To promote and understand privacy and compliance related risk, ensuring formal acceptance and awareness between executives and to address security and privacy within vendor SLAs.</td>
<td>To drive continual improvement efforts, proactively develop relevant audit documentation and ensure vendor compliance with the organization’s software development policy.</td>
</tr>
<tr>
<td>Activities</td>
<td>• Unify regulatory pressures. • Identify PII Obligations • Create Policy</td>
<td>• Identify PII Data Inventory • Require Security Sign-Off for Compliance-Related Risk • Implement and Track Controls for Compliance • Paper All Vendor Contracts with Software Security SLAs • Ensure Executive Awareness of Compliance and Privacy Obligations</td>
<td>• Create Regulatory Eye-Candy • Impose Policy on Vendors • Drive Feedback form SSDL Data Back to Policy</td>
</tr>
<tr>
<td>Audit Considerations</td>
<td>• Has a repository of applicable legislation and regulation been developed? • Have compliance requirements been mapped? • Are applicable privacy requirements been identified and documented? • Is a Software Security Policy in place? • Does the policy address compliance requirements?</td>
<td>• Are all PII data points and processing systems documented? • Do executives understand compliance and privacy obligations? • Do vendor SLAs address Privacy and Security? • Are software related risks governed by a formal risk acceptance process?</td>
<td>• Has the organization identified relevant compliance evidence? • Is compliance with the Organization’s policy mandated in contracts? • Is third-party compliance assured? • Is the software development policy updated following feedback from the SSDL Process?</td>
</tr>
</tbody>
</table>
Training

Training has always played a critical role in software security because software developers and architects often start with very little security knowledge.

<table>
<thead>
<tr>
<th>Training (T)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maturity Level</strong></td>
</tr>
<tr>
<td><strong>Objective</strong></td>
</tr>
<tr>
<td><strong>Activities</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Audit Considerations</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
## Intelligence

### Attack Models

Attack Models capture information used to think like an attacker: threat modelling, abuse case development and refinement, data classification, and technology-specific attack patterns.

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>AM1</th>
<th>AM2</th>
<th>AM3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td>To ensure that the value of data is properly assessed to both the organization and potential attackers in addition to identifying interested malicious actors, attack types and vulnerabilities currently being exploited.</td>
<td>To ensure the ongoing consideration of attack patterns and abuse cases for software deployments and to raise awareness and understanding of threats and threat actors.</td>
<td>To ensure that the organization continues to consider the approach and impacts of new attack methods while also acquiring or developing automated tools to aid testers to perform adversarial testing.</td>
</tr>
</tbody>
</table>
| **Activities**  | • Create a Data Classification Scheme and Inventory  
• Identify Potential Attackers  
• Gather and Use Attack Intelligence | • Build Attack Patterns and Abuse Cases Tied to Potential Attackers  
• Create Technology-Specific Attack Patterns  
• Build and Maintain a Top N Possible Attacks List  
• Collect and Publish Attack Stories  
• Build an Internal Forum to Discuss Attacks | • Have a Science Team that Develops New Attack Methods  
• Create and Use Automation to do what Attackers will do |
| **Audit Considerations** | • Has the organization developed or followed a data classification scheme?  
• Have all software applications been inventoried based upon the classification of data processed?  
• Does the application inventory consider off-premise software applications?  
• Has the organization completed a Threat Actor Assessment to identify potential attackers, motivations and capabilities?  
• Have Threat Actor Profiles been developed for all identified potential attackers?  
• Do the profiles of noteworthy attackers contain extensive detail enhanced by third-party resources? | Does the organization document the attack patterns of identified potential attackers and for specific technologies?  
Are abuse cases developed for testing activities?  
Is a list maintained of the most likely attacks?  
Does the list contain observed attack data, industry trends, third-party services other online resources?  
Is historical attack data shared internally for awareness and education?  
Is there an internal forum dedicated to discussing attacker trends, new vulnerabilities and information concerning publicly advertised incidents?  
Is the forum attended by the SSG, the satellite and other interested parties? | • Does the organization have a team who’s responsible for developing new attack methods?  
• Does the team look beyond known types of weaknesses and focus on developing innovative attack methods?  
• Does the team present newly identified attack methods at conferences?  
• Does the team develop tools for testing attacks?  
• Does the team seek to improve testing from an attacker’s perspective through developing automated testing tools? |
Security Features and Design
The Security Features & Design practice is charged with creating usable security patterns for major security controls (meeting the standards defined in the Standards & Requirements practice), building middleware frameworks for those controls, and creating and publishing other proactive security guidance.

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>SFD1</th>
<th>SFD2</th>
<th>SFD3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>To ensure the continuing consideration of software security amongst architecture groups and support project teams through the provision of pre-approved security features for common or solution designs.</td>
<td>To ensure that capability is developed to enable the development of architectures that solve difficult design problems and to ensure that middleware frameworks or common libraries are secure-by-design.</td>
<td>To ensure that approved security features and designs are maintained and published so that they remain available for reuse by project teams.</td>
</tr>
</tbody>
</table>
| Activities      | • Build and Publish Security Features  
• Engage the Software Security Group (SSG) with Architecture | • Build Secure-By-Design Middleware Frameworks and Common Libraries  
• Create SSG Capability to Solve Difficult Design Problems | • Form a Review Board or Central Committee to Approve and Maintain Secure Design Patterns  
• Require Use of Approved Security Features and Frameworks  
• Find and Publish Mature Design Patterns from the Organization |
| Audit Considerations | • Does the organization proactive provide guidance concerning required security features to common or complex design problems?  
• Is information concerning the required security features published so that project teams may reuse requirements?  
• Do project teams implement published security features into design?  
• Is the SSG integrated with the architecture community?  
• Do the architecture community take responsibility for security?  
• Does the architecture community assist in the development of secure designs and include software security into design standards? | • Does the SSG develop and promote secure-by-design middleware frameworks and common libraries?  
• Are identified frameworks and libraries used by project teams?  
• Is the adoption or implementation of secure frameworks and libraries recommended during architecture analysis?  
• Is the SSG engaged during the design and requirements gathering phase of project development?  
• Is the SSG involved in the review security designs? | • Is a review board established to approve and maintain secure design patterns?  
• Does the review board periodically review and update design patterns?  
• Is the adoption of new technology considered in conjunction with the SSG?  
• Are design patterns published for stakeholders to use?  
• Do stakeholders know where to find approved design patterns?  
• Are projects required to use approved security features, designs and frameworks?  
• Are projects prevented from progressing if suitable features, designs and frameworks are ignored? |
Standards and Requirements

The Standards & Requirements practice involves eliciting explicit security requirements from the organization, determining which COTS to recommend, building standards for major security controls (such as authentication, input validation, and so on), creating security standards for technologies in use, and creating a standards review board.

<table>
<thead>
<tr>
<th>Standards and Requirements (SR)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maturity Level</strong></td>
</tr>
<tr>
<td>Objective</td>
</tr>
<tr>
<td>Activities</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Audit Considerations</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
SSDL Touchpoints

Architecture Analysis

Architecture Analysis encompasses capturing software architecture in concise diagrams, applying lists of risks and threats, adopting a process for review (such as STRIDE or Architecture Risk Analysis), and building an assessment and remediation plan for the organization.

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>AA1</th>
<th>AA2</th>
<th>AA3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>To ensure that all software receives a review of security features and that design reviews are performed for high-risk applications following application risk assessments.</td>
<td>To ensure that a standardized is maintained for the description of architecture analysis, its components and support project teams to perform self-assessments through mentoring.</td>
<td>To ensure that architecture analysis results are fed back to the security design committee for improving designs and that design review efforts are led by software architects.</td>
</tr>
</tbody>
</table>
| Activities     | • Perform Security Feature Review  
• Perform Design Review for High-Risk Applications  
• Have SSG Lead Design Review Efforts  
• Use a Risk Questionnaire to Rank Applications | • Define and Use Architecture Analysis (AA) process  
• Standardize Architectural Descriptions (Including Data Flow)  
• Make the SSG Available as a AA Resource or Mentor | • Have Software Architects Lead Design Review Efforts  
• Drive Analysis Results into Standard Architecture Patterns |
| Audit Considerations | • Is there a process in place to review the security features of a solution?  
• Does the security features review consider the design and implementation of the security feature?  
• Is design review prioritized based upon risk assessment?  
• Are risk questionnaires checked for accuracy and validity?  
• Is detailed design review completed for high-risk application?  
• Do design review check against known security risks?  
• Do design reviews result in documented architecture flaws and mitigation plans?  
• Are review efforts led by the SSG?  
• Are Architects and Implementers involved in the design review?  
• Are consultants engage for challenging design review activities? | • Has an architecture analysis process been defined and documented?  
• Is the process strictly followed to standardize the approach to design reviews?  
• Is the documentation reviewed and updated over time?  
• Has a standardized vocabulary been defined for architecture analysis activities?  
• Is the documentation understood by non-security experts?  
• Do project teams perform architecture analysis?  
• Do project teams specifically analyze design elements for security features? | Do Software Architects lead design review efforts?  
Does the review process incorporate data-level analysis?  
Does a minimum-security baseline exist for design review results?  
Are failures fed back to the design committee? |
The Code Review practice includes use of code review tools, development of tailored rules, customized profiles for tool use by different roles (for example, developers versus auditors), manual analysis, and tracking/measuring results.

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>CR1</th>
<th>CR2</th>
<th>CR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>To ensure that all software projects are reviewed at a code level through a combination of automated and manual assessment in addition to ensuring that identified bugs are incorporated into reporting processes and software development training materials.</td>
<td>To ensure that testing focuses upon eliminating the most important kind of bugs from applications and to minimize wasted efforts through automation with tool mentoring and customization.</td>
<td>To ensure that all assessment results are communicated through a single reporting and remediation process, in addition to ensuring that coding standards are followed and that identified bugs are addressed during development across the entire codebase.</td>
</tr>
</tbody>
</table>
| Activities     | • Have the Software Security Group (SSG) Perform Ad-Hoc Review  
• Use Automated Tools along with Manual Review  
• Make Code Review Mandatory for all Projects  
• Use Centralized reporting to Close the Knowledge Loop and Drive Training | • Assign Tool Mentors  
• Use Automated Tools with Tailored Rules  
• Use a Top N Bugs list | • Build a Factory  
• Build a Capability for Eradication Specific Bugs from the Entire Codebase  
• Automate Malicious Code Detection  
• Enforce Coding Standards |
| Audit Considerations | • Does the software Security Group (SSG) perform ad-hoc reviews of high-risk applications?  
• Is static analysis of code performed in addition to manual reviews?  
• Is code review mandated for all projects?  
• Is the automated review of high-risk projects supplemented with manual review?  
• Are minimum acceptable standards defined for code review processes?  
• Do projects that fail to meet defined minimum standards get returned for fix and re-evaluation before progressing to operations?  
• Are identified bugs tracked in a centralized repository?  
• Are summary and trend reports generated?  
• Is bug and bug report data incorporated back into training resources?  

• Are mentors available to assist developers to review application code?  
• Are code review tools customized to reduce false positives?  
• Is the solution configured to avoid irrelevant checks?  
• Has a list of top bugs been defined?  
• Is the list developed using real data gathered from code review, testing and actual incidents?  
• Is the list updated regularly?  

• Are multiple analysis results reported and remediated through a single process?  
• When new bugs are identified is a search conducted across the entire codebase to identify additional instances?  
• Are identified bugs addressed during development and ahead of formal code review activities?  
• Are automated solutions implemented to identify malicious code?  
• Is the malicious code detection tool customized to codify acceptable and unacceptable code patterns?  
• Are the organization’s coding standards enforced?  
• Is code rejected for failing to comply with coding standards? |
Security Testing

The Security Testing practice is concerned with pre-release testing, including integrating security into standard quality assurance processes. The practice includes use of black box security tools (including fuzz testing) as a smoke test in QA, risk-driven white box testing, application of the attack model, and code coverage analysis. Security testing focuses on vulnerabilities in construction.

<table>
<thead>
<tr>
<th>Security Testing (ST)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maturity Level</strong></td>
</tr>
<tr>
<td><strong>Objective</strong></td>
</tr>
</tbody>
</table>
| **Activities**        | • Ensure QA Supports Edge/Boundary Value Condition Testing  
• Drive Tests with Security Requirements and Security Features | • Integrate Black Box Security Tools into the QA Process  
• Share Security Results with QA  
• Perform Fuzz Testing Customized to Application APIs | • Drive Tests with Risk Analysis Results  
• Leverage Coverage Analysis  
• Begin to Build and Apply Adversarial Security Tests (Abuse Cases) |
| **Audit Considerations** | • Does the QA team perform basic adversarial tests?  
• Do basic adversarial tests include boundary conditions and edge cases?  
• Does QA testing assess the declarative security mechanisms derived from requirements and security features? | • Does the organization use black-box testing tools to supplement QA tests?  
• Do the tools cover web applications and network protocols?  
• Is the SSG involved in helping to implement the tools and interpret results?  
• Are the results of security reviews shared with the QA team?  
• Is security testing performed as part of regressions tests?  
• Are security tests driven by abuse cases and tweaks to functional tests?  
• Does the organization implement a fuzzing framework?  
• Is the fuzzing framework customized for the organization’s application interfaces? | • Are testing activities guided by the results of architecture analysis?  
• Are adversarial tests priorities by risk profile?  
• Do testing efforts consider the range of capabilities and motivations of potential attackers?  
• Do testing efforts assess the coverage of code exercised by automated testing?  
• Is omitted code reviewed and tested manually?  
• Is testing performed based upon function, line or multiple condition coverage?  
• Does testing incorporate previously identified abuse and misuse cases? |
Deployment
Penetration Testing

The Penetration Testing practice involves standard outside-in testing of the sort carried out by security specialists. Penetration testing focuses on vulnerabilities in the final configuration and provides direct feeds to defect management and mitigation.

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Penetration Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The Penetration Testing practice involves standard outside-in testing of the sort carried out by security specialists. Penetration testing focuses on vulnerabilities in the final configuration and provides direct feeds to defect management and mitigation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Penetration Testing (PT)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Maturity Level</strong></td>
</tr>
<tr>
<td><strong>Objective</strong></td>
</tr>
<tr>
<td><strong>Activities</strong></td>
</tr>
<tr>
<td><strong>Audit Considerations</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MINISTRY OF TRANSPORT AND COMMUNICATIONS</th>
<th>Q-CERT</th>
<th>MINISTRY OF TRANSPORT AND COMMUNICATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title:</strong> Software Security and Quality Assurance (SSQA) Framework Guidance</td>
<td><strong>Version:</strong> 1.0</td>
<td><strong>Classification:</strong> PUBLIC</td>
</tr>
<tr>
<td><strong>Page 24 of 59</strong></td>
<td><strong>Q-CERT</strong></td>
<td><strong>Page 24 of 59</strong></td>
</tr>
</tbody>
</table>
Software Environment

The Software Environment practice concerns itself with OS and platform patching, web application firewalls, installation and configuration documentation, application monitoring, change management and, ultimately, code signing.

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>SE1</th>
<th>SE2</th>
<th>SE3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>To ensure that host and network infrastructure is in place to support software deployments and to monitor application inputs through manual or automated measures.</td>
<td>To ensure that deployment teams are provided with appropriate guidance for the secure installation and configuration of software and equipped with the means to identify application tampering.</td>
<td>To protect intellectual property, identify misbehavior or attempts to attack software deployments and to isolate solutions to ease deployment and reduce risk in the event of attack.</td>
</tr>
</tbody>
</table>
| Activities | - Use Application Input Monitoring  
- Ensure Host and Network Security Basics are in Place | - Publish Installation Guides  
- Use Code Signing | - Use Code Protection  
- Use Application Behavior Monitoring and Diagnostics  
- Use Application Containers |
| Audit Considerations | - Does the organization monitor the inputs to software solutions?  
- Are logs actively or periodically monitored?  
- Is the host system (and network) configured to disable unnecessary services?  
- Is there a patching policy in place?  
- Are patches being applied within a reasonable timeframe?  
- Are firewall rules configured and reviewed regularly?  
- Are administrative accounts suitably secured? | - Have deployment guides been developed to direct installation and configuration activities?  
- Do deployment guides outline specific steps for addressing security?  
- Does the deployment guides discuss Commercial-Off-The-Shelf (COTS) components  
- Are automated aspects of deployment understandable to operators?  
- Is code signing implemented to validate the authenticity and integrity of software packages?  
- Is code signing used for shrink-wrapped applications and thick clients? | - Does the application utilize obfuscation techniques to protect intellectual property and inhibit reverse engineering attempts?  
- Are controls employed as part of the production build and release process?  
- Is the obfuscation technique employed for variable data?  
- Are platform specific obfuscation techniques used?  
- Does the organization utilize controls to monitor software behavior for signs of misuse or attack?  
- Does behavior monitoring consider interaction with the host and data created, consumed or manipulated?  
- Does the organization utilize application containers?  
- Do containers implement any security controls? |
Configuration Management and Vulnerability Management

The Configuration Management & Vulnerability Management practice concerns itself with patching and updating applications, version control, defect tracking and remediation, and incident handling.

## Configuration Management and Vulnerability Management (CMVM)

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>CMVM1</th>
<th>CMVM2</th>
<th>CMVM3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td>To ensure the integration of projects and the SSG with incident response capabilities to enable awareness and capabilities for responding to software security incidents as well as to ensure that identified weaknesses or defects are addressed in development activities.</td>
<td>To understand the extent of software deployments enabling the swift changes and patches to code that is under attack or compromised and to track bugs from identification through to resolution.</td>
<td>To encourage the responsible disclosure of vulnerabilities and the simulation of incidents or crises arising from exploitation and to remediate all instances of identified bugs and prevent their reintroduction during future developments.</td>
</tr>
</tbody>
</table>
| **Activities**  | • Create or Interface with Incident Response  
• Identify Software Defects Found in Operations Monitoring and feed them Back to Development | • Have Emergency Codebase Response  
• Track Software Bugs Found in Operations Through the Fix Process  
• Develop an Operations Inventory of Applications | • Fix All Occurrences of Software Bugs Found in Operations  
• Enhance the SSDL to prevent software bugs found in operations  
• Simulate software crisis  
• Operate a bug bounty program |
| **Audit Considerations**  | • Does the organization have a designated security response team?  
• Do projects and the SSG have a designated point of contact for security issues and incidents?  
• Are project teams and the SSG aware of their security point(s) of contact and the response team?  
• Does the organization utilize a consistent process for incident reporting and handling?  
• Does the SSG regularly meet with the incident response team and project managers to keep-up-to-date with issues or concerns?  
• Does the Organization leverage automation to check application and environment?  
• Are defects and weaknesses reported back to development teams? | • Does the organization have specific incident response plans addressing the compromise or attack of software solutions?  
• Do Software security incident response plans support the rapid response to code-level weaknesses?  
• Are defects and weaknesses tracked from discovery and notification through to remediation?  
• Does the organization maintain an inventory of software applications in operation to efficiently address weaknesses in the event of discovery or attack? | • Are Incidents inspected for root causes to generate further recommendations?  
• Is bug and weaknesses data fed back into development activities to prevent future reoccurrence?  
• Do Projects consistently collect, and report data and metrics related to vulnerabilities and incidents? Do the development address all instances of identified bugs or weaknesses when notified?  
• Is the incident response process tested regularly against plans for software security incidents?  
• Does the organization operate a bug bounty program? |
What is a Software Security Group (SSG)?
The Software Security Group (SSG) is an internal group charged with carrying out and facilitating software security.

The best SSG members are software security people, but software security people can sometimes be difficult to find and can often be complemented by individuals of different backgrounds. For this reason, SSGs come in a variety of shapes and sizes and often the most effective SSGs include both people with deep coding experience and people from architectural backgrounds.

Given the vast domains of knowledge and experience within the SSG, the group is often well placed to mentor, train, and work directly with developers. This requires that at least a portion of the SSG staff are equipped with good communications skills and teaching capability which may need to be developed through training activities.

If software security expertise needs to be created from scratch, start with developers and teach them about security.
Starting with network security people and attempting to teach them about software specifics often fails to produce the desired results.
Unfortunately, no amount of traditional security knowledge can overcome a lack of experience building software.

And remember, while it may seem plausible that two roles such as software architecture and code review share a common skillset, the best code reviewers sometimes make very poor software architects.
It is therefore important to ensure a depth and breadth of capabilities across the SSG.

What is meant by a Satellite?
A satellite, is a group of individuals (often developers, testers, and architects) who share a basic interest in software security but are not directly employed in the SSG.

Sometimes the satellite is widely distributed, with one or two members in each product group.
Sometimes the satellite is more focused and gets together regularly to compare notes, learn new technologies, and expand the understanding of software security in an organization.

Identifying and fostering a strong satellite is important to the success of many software security initiatives and some BSIMM activities target the satellite explicitly.
Roles and Responsibilities

Involved in the development and assurance of software solutions is a variety of stakeholders and roles, each with specific responsibilities.

These role holders, responsible from management through to operations ensure a breadth of oversight and influence to support the alignment of software development with best practice approaches for software security.

Developers (Individuals performing detailed design and implementation of the software)
Must practice security engineering, ensuring that the systems that we build are defensible, as well as free of known or foreseeable vulnerabilities and weaknesses. As secure software development maturity grows, developers should become increasingly responsible for conducting a large number of the SSQA activities themselves with the SSG helping in special cases and providing oversight.

Architects (Individuals performing high-level design work and large-scale system engineering)
Must practice security engineering, ensuring that the systems that we build are defensible, as well as free of known or foreseeable vulnerabilities and weaknesses. As secure software development maturity grows, architects should become increasingly responsible for conducting design review activities themselves with the SSG helping in special cases and providing oversight.

Managers (Individuals performing day-to-day management of development staff and Business or Product Owners)
Must understand how early investment in security design and security analysis affects the degree to which users will trust their products.
Business requirements should explicitly address security needs.

Testers
Are concerned with routine testing and verification assessments during gate-based assessment activities. Testers should communicate weaknesses and bugs, found during testing, to design and development teams so they can be addressed earlier in the development lifecycle to prevent their reintroduction.

Satellite
As a group of developers, architects, software managers, testers, and similar roles with natural affinity or interest for software security, the Satellite is responsible for assisting the Software Security Group (SSG) in the fulfilment of the Software Security Initiative (SSI) and for promoting software security within their respective business functions.

Software Security Group (SSG)
The internal group charged with organizing the Satellite and facilitating the Software Security Initiative (SSI).

Administrators
Must understand the distributed nature of modern systems and begin to practice the principle of least privilege, especially when it comes to the applications they host or attach to as services in the cloud.
Operations
People must continue to design reasonable networks, defend them, and keep them working. Software security doesn't end when software is “shipped”.

Vendors (including those who supply Commercial-Off-The-Shelf (COTS), custom software and software-as-a-service)
Individuals or organizations engaged in providing solutions, systems or services should be held responsible for complying with defined software security standards and Service Level Agreements (SLAs) governing software security.
Queries, Concerns, Feedback and Further Information

Assessing and aligning with the Software Security and Quality Assurance (SSQA) framework can initially be considered daunting and as with all change, prompt many questions. It is recognized that not every answer can be provided in a single document, although as time passes the SSQA Framework Guidance will be updated to provide information based upon feedback.

Therefore, if any questions remain that have not been answered within this document, or if there are any ways in which this document (or the wider framework) can be enhanced to address any knowledge gaps or concerns, it is requested that these are forwarded to the Compliance & Information Protection Department (CIPD) via e-mail.
Appendix: SSQA Domains, Practices and Activities

GOVERNANCE: Strategy & Metrics (SM)

The Strategy & Metrics practice encompasses planning, assigning roles and responsibilities, identifying software security goals, determining budgets, and identifying metrics and gates.

SM LEVEL 1

[SM1.1] Publish process (roles, responsibilities, and plan), evolve as necessary.

The process for addressing software security is broadcast to all stakeholders so that everyone knows the plan. Goals, roles, responsibilities, and activities are explicitly defined. Most organizations pick and choose from a published methodology, such as the Microsoft SDL or the Cigital Touchpoints, and then tailor the methodology to their needs. An SSDL process must be adapted to the specifics of the development process it governs (e.g., waterfall, agile, etc.) and evolves both with the organization and the security landscape. A process must be published to count. In many cases, the methodology is published only internally and is controlled by the SSG. The SSDL does not need to be publicly promoted outside of the firm to have the desired impact.

[SM1.2] Create evangelism role and perform internal marketing.

In order to build support for software security throughout the organization, someone in the SSG plays an evangelism role. This internal marketing function helps keep executives and all other stakeholders current on the magnitude of the software security problem and the elements of its solution. Evangelists might give talks for internal groups including executives, extend invitations to outside speakers, author white papers for internal consumption, or create a collection of papers, books, and other resources on an internal website and promote its use. Ad hoc conversations between the SSG and executives, or an SSG where “everyone is an evangelist,” do not achieve the desired results. A canonical example of such an evangelist was Michael Howard’s role at Microsoft just after the Gates memo. As another example, an agile coach familiar with security can help teams adopt better software security practices as they transform to an agile methodology.

[SM1.3] Educate executives.

Executives are periodically shown the consequences of inadequate software security and the negative business impact that poor security can have. They’re also shown what other organizations are doing to attain software security. By understanding both the problem and its proper resolution, executives come to support the software security initiative as a risk management necessity. In its most dangerous form, such education arrives courtesy of malicious hackers or public data exposure incidents. Preferably, the SSG demonstrates a worst-case scenario in a controlled environment with the permission of all involved (e.g., actually showing working exploits and their business impact). In some cases, presentation to the Board can help garner resources for an ongoing software security initiative. Bringing in an outside guru is often helpful when seeking to bolster executive attention. Tying education to specific development areas, such as mobile or cloud services, can help convince leadership to accept SSG recommendations where they might otherwise be ignored in favor of faster release dates or other priorities.

[SM1.4] Identify gate locations, gather necessary artefacts.

The software security process includes release gates/checkpoints/milestones at one or more points in the SDLC or, more likely, the SDL Cs. The first two steps toward establishing security-specific release gates are: 1) to identify gate locations that are compatible with existing development practices and 2) to begin gathering the input necessary for making a go/no-go decision. Importantly at this stage, the gates are not enforced.

For example, the SSG can collect security testing results for each project prior to release but stop short of passing judgment on what constitutes sufficient testing or acceptable test results. Shorter release cycles, as are seen in organizations practicing agile development, often require creative approaches to collecting the
right evidence and rely heavily on lightweight, super-fast automation. The idea of identifying gates first and only enforcing them later is extremely helpful in moving development toward software security without major pain. Socialize the gates, and only turn those on once most projects already know how to succeed. This gradual approach serves to motivate good behavior without requiring it.

**SM LEVEL 2**

**[SM2.1] Publish data about software security internally.**
The SSG publishes data internally on the state of software security within the organization to facilitate improvement. The information might come as a dashboard with metrics for executives and software development management. Sometimes, publication is not shared with everyone in a firm, but rather with the relevant executives only. In this case, publishing information up to executives who then drive change in the organization is necessary. In other cases, open book management and publishing data to all stakeholders helps everyone know what’s going on, with the philosophy that sunlight is the best disinfectant. If the organization’s culture promotes internal competition between groups, this information adds a security dimension to the game. The time compression associated with agile development calls for measurements that can be taken quickly and accurately, focusing less on historical trends (e.g., bugs per release) and more on speed (e.g., time to fix).

**[SM2.2] Enforce gates with measurements and track exceptions.**
SDLC security gates are now enforced: in order to pass a gate, a project must either meet an established measure or obtain a waiver. Even recalcitrant project teams must now play along. The SSG tracks exceptions. A gate could require a project to undergo code review and remediate any critical findings before release. In some cases, gates are directly associated with controls required by regulations, contractual agreements, and other business obligations, and exceptions are tracked as required by statutory or regulatory drivers. In other cases, gate measures yield key performance indicators that are used to govern the process. A revolving door or a rubber stamp exception process does not count. If some projects are automatically passed, that defeats the purpose of enforcing gates. Even seemingly innocuous development projects, such as a new mobile client for an existing back-end or porting an application to a cloud environment from an internal data center, must successfully pass the prescribed security gates in order to progress.

**[SM2.3] Create or grow a satellite.**
The satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or skill. Identifying this group is a step towards creating a social network that speeds the adoption of security into software development. One way to begin is to track the people who stand out during introductory training courses. Another way is to ask for volunteers. In a more top-down approach, initial satellite membership is assigned to ensure complete coverage of all development/product groups. Ongoing membership should be based on actual performance. A strong satellite is a good sign of a mature software security initiative. In new or fast-moving technology areas such as mobile development, or development paradigms such as DevOps, satellite members can help combine software security skills with domain knowledge that might be underrepresented in the SSG. Agile coaches make particularly useful satellite members.

**[SM2.5] Identify metrics and use them to drive budgets.**
The SSG and its management choose the metrics that define and measure software security initiative progress. These metrics will drive the initiative’s budget and allocation of resources, so simple counts and statistics won’t suffice. Metrics also allow the SSG to explain its goals and its progress in quantitative terms. One such metric could be security defect density. A reduction in security defect density could be used to show a decreasing cost of remediation over time. Recall that in agile methodologies, metrics are best collected early and often in a lightweight manner. The key here is to tie technical results to business objectives in a clear and
obvious fashion in order to justify funding. Because the concept of security is already tenuous to business people, making this explicit tie can be very helpful.

**[SM2.6] Require security sign-off.**
The organization has an initiative-wide process for accepting security risk and documenting accountability. A risk acceptor signs off on the state of all software prior to release. For example, the sign-off policy might require the head of the business unit to sign off on critical vulnerabilities that have not been mitigated or SSDL steps that have been skipped. The policy must apply to outsourced projects, such as a boutique mobile application, and to projects that will be deployed in external environments, such as the cloud. Informal or uninformed risk acceptance alone does not count as security sign off, as the act of accepting risk is more effective when it is formalized (e.g., with a signature, form submission, or something similar) and captured for future reference. Similarly, simply stating that certain projects never need a sign-off does not achieve the desired results.

**SM LEVEL 3**

**[SM3.1] Use an internal tracking application with portfolio view.**
The SSG uses a centralized tracking application to chart the progress of every piece of software in its purview, regardless of development methodology. The application records the security activities scheduled, in progress, and completed. It incorporates results from activities such as architecture analysis, code review, and security testing even when they happen in a tight loop. The SSG uses the tracking application to generate portfolio reports for many of the metrics it uses. A combined inventory and risk posture view is fundamental. In many cases, these data are published at least among executives. Depending on the culture, this can cause interesting effects through internal competition. As an initiative matures and activities become more distributed, the SSG uses the centralized reporting system to keep track of all of the moving parts.

**[SM3.2: 9] Run an external marketing program.**
The SSG helps the firm market the software security initiative outside to build external support. Software security grows beyond being a risk reduction exercise and becomes a competitive advantage or market differentiator.
The SSG might write papers or books about its SSDL. It might have a public blog. It might participate in external conferences or trade shows. In some cases, a complete SSDL methodology can be published and promoted externally. Mobile and cloud security projects can make great software security case studies.

Sharing details externally and inviting critique can bring new perspectives into the firm.
GOVERNANCE: Compliance & Policy (CP)

The Compliance & Policy practice is focused on identifying controls for compliance regimens such as PCI-DSS and HIPAA, developing contractual controls such as service level agreements to help control COTS software risk, setting organizational software security policy, and auditing against that policy.

CP LEVEL 1

[CP1.1] Unify regulatory pressures.
If the business or its customers are subject to regulatory or compliance drivers such as FFIEC, GLBA, OCC, PCI-DSS, SOX, HIPAA, or others, the SSG acts as a focal point for understanding the constraints such drivers impose on software. In some cases, the SSG creates a unified approach that removes redundancy and conflicts from overlapping compliance requirements. A formal approach will map applicable portions of regulations to control statements explaining how the organization complies. As an alternative, existing business processes run by legal or other risk and compliance groups outside the SSG could also serve as the regulatory focal point. The goal of this activity is to create one set of software security guidance so that compliance work is completed as efficiently as possible. Some firms move on to guide exposure by becoming directly involved in standards groups exploring new technologies in order to influence the regulatory environment.

[CP1.2] Identify PII obligations.
The way software handles personally identifiable information (PII) could be explicitly regulated, but even if it isn’t, privacy is a hot topic. The SSG plays a key role in identifying and describing PII obligations stemming from regulation and customer expectations. It uses this information to promote best practices related to privacy. For example, if the organization processes credit card transactions, the SSG will identify the constraints that the PCI DSS places on the handling of cardholder data and inform all stakeholders. Note that outsourcing to hosted environments (e.g. the cloud) does not relax PII obligations. Also note, firms that create software products that process PII (but who don’t necessarily handle PII directly) can get credit by providing privacy controls and guidance for their customers.

Because it’s easy to lose your phone, mobile devices add new PII security considerations.

[CP1.3] Create policy.
The SSG guides the rest of the organization by creating or contributing to software security policy that satisfies regulatory and customer-driven security requirements. The policy provides a unified approach for satisfying the (potentially lengthy) list of security drivers at the governance level. As a result, project teams can avoid keeping up with the details involved in complying with all applicable regulations. Likewise, project teams don’t need to re-learn customer security requirements on their own. The SSG policy documents are sometimes focused around major compliance topics such as the handling of PII or the use of cryptography. In some cases, policy documents relate directly to the SSDL and its use in the firm. Because they’re so new, codifying decisions about cloud and mobile architectures can add some much-needed pizazz to the policy discussion. Architecture standards and coding guidelines are not examples of software security policy. On the other hand, policy that prescribes and mandates the use of coding guidelines and architecture standards for certain categories of applications does count. Policy is what is permitted and denied at the initiative level. If it’s not mandatory, it’s not policy.
CP LEVEL 2

[CP2.1] Identify PII data inventory.
The organization identifies the kinds of PII processed or stored by each of its systems and their data repositories, including mobile and cloud environments. A PII inventory can be approached in two ways: starting with each individual application by noting its PII use or starting with particular types of PII and the applications that touch them. In either case, an inventory of data repositories is required. Note that when applications are distributed across multiple deployment environments, PII inventory control can get tricky. Do not ignore it. When combined with the organization’s PII obligations, this inventory guides privacy planning. For example, the SSG can now create a list of databases that would require customer notification if breached.

[CP2.2] Require security sign-off for compliance-related risk.
The organization has a formal compliance risk acceptance and accountability process addressing all software development projects, regardless of development methodology. The SSG might act as an advisor when the risk acceptor signs off on the state of the software prior to release. For example, the sign-off policy might require the head of the business unit to sign off on compliance issues that have not been mitigated or SSDL steps related to compliance that have been skipped. Signoff should be explicit and captured for future reference. Any exceptions should be tracked, even under the fastest of agile methodologies. An application without security defects might still be non-compliant.

[CP2.3] Implement and track controls for compliance.
The organization can demonstrate compliance with applicable regulations because its SSDL is aligned with the control statements developed by the SSG (see [CP1.1 Unify regulatory pressures]). The SSG tracks the controls, shepherds problem areas, and makes sure auditors and regulators are satisfied. If the organization’s SDLC is predictable and reliable, the SSG might be able to largely sit back and keep score. If the SDLC is uneven, less reliable, or in a big, big hurry, the SSG could be forced to take a more active role as referee. A firm doing this properly can explicitly associate satisfying its compliance concerns to following its SSDL.

[CP2.4] Paper all vendor contracts with software security SLAs.
Vendor contracts include a service-level agreement (SLA) ensuring that the vendor will not jeopardize the organization’s compliance story and software security initiative. This is particularly important when controlling cloud computing providers. Each new or renewed contract contains a set of provisions requiring the vendor to address software security and deliver a product or service compatible with the organization’s security policy. In some cases, open source licensing concerns initiate the vendor control process. That can open the door for further software security language in the SLA. Traditional IT security requirements and a simple agreement to allow penetration testing are not sufficient.

[CP2.5: 38] Ensure executive awareness of compliance and privacy obligations.
The SSG gains executive buy-in around compliance and privacy activities. Executives are provided plain-language explanations of the organization’s compliance and privacy obligations and the potential consequences for failing to meet those obligations. For some organizations, explaining the direct cost and likely fallout from a data breach could be an effective way to broach the subject. For other organizations, having an outside expert address the Board works because some executives value outside perspective more than internal perspective. One sure sign of proper executive awareness is adequate allocation of resources to get the job done. Be aware that the light and heat typically following a breach will not last.
[CP3.1] Create regulator eye candy.
The SSG has the information regulators want. A combination of written policy, controls documentation, and artefacts gathered through the SSDL gives the SSG the ability to demonstrate the organization’s compliance story without a fire drill for every audit or a piece of paper for every sprint. In some cases, regulators, auditors, and senior management are satisfied with the same kinds of reports, which may be generated directly from various tools.

[CP3.2] Impose policy on vendors.
Vendors are required to adhere to the same policies used internally. Vendors must submit evidence that their software security practices pass muster. This goes for cloud and mobile platform providers as well. Evidence could include results from code reviews or penetration tests. Vendors may also attest to the fact that they are carrying out certain SSDL processes. In some cases, a BSIMM score is used to help ensure that vendors are complying with the firm’s policies.

[CP3.3] Drive feedback from SSDL data back to policy.
Information from the SSDL is routinely fed back into the policy creation process. Policies are improved to find defects earlier or prevent them from occurring in the first place. Blind spots are eliminated based on trends in SSDL failures. For example, inadequate architecture analysis, recurring vulnerabilities, ignored security gates, or choosing the wrong firm to carry out a penetration test can expose policy weakness. Likewise, policies that impose too much bureaucracy might need to be adjusted to fit agile methodologies. Over time, policies should become more practical and easier to carry out. Ultimately, policies align themselves with the SSDL data and enhance and improve a firm’s effectiveness.
GOVERNANCE: Training (T)
Training has always played a critical role in software security because software developers and architects often start with very little security knowledge.

T LEVEL 1
[T1.1] Provide awareness training.
The SSG provides awareness training in order to promote a culture of software security throughout the organization.
Training might be delivered by SSG members, by an outside firm, by the internal training organization, or through eLearning. Course content isn’t necessarily tailored for a specific audience. For example, all programmers, quality assurance engineers, and project managers could attend the same 'Introduction to Software Security' course. This common activity can be enhanced with a tailored approach to a course that addresses a firm’s culture explicitly.
Generic introductory courses covering basic IT security and high-level software security concepts do not generate satisfactory results. Likewise, providing awareness training only to developers and not to other roles is also insufficient.

[T1.5] Deliver role-specific advanced curriculum (tools, technology stacks, and bug parade).
Software security training goes beyond building awareness and enables trainees to incorporate security practices into their work. The training is tailored to the role of trainees; trainees get information about the tools, technology stacks, development methodologies, or kinds of bugs that are most relevant to them. An organization might offer four tracks for engineers: one for architects, one for Java developers, one for mobile developers, and a fourth for testers. Tool-specific training is also commonly observed in a curriculum. Don’t forget that training will be useful for many different roles in an organization, including QA, product management, executives, and others.

[T1.6] Create and use material specific to company history.
In order to make a strong and lasting change in behavior, training includes material specific to the company’s history. When participants can see themselves in the problem, they are more likely to understand how the material is relevant to their work and to know when and how to apply what they have learned. One way to do this is to use noteworthy attacks on the company as examples in the training curriculum. Be wary of training that covers platforms not used by developers (Windows developers don’t care about old Unix problems) or examples of problems only relevant to languages no longer in common use (Java developers don’t need to understand buffer overflows in C). Stories from company history can help steer training in the right direction only if the stories are still relevant and not overly censored.

[T1.7] Deliver on-demand individual training.
The organization lowers the burden on trainees and reduces the cost of delivering training by offering on-demand training for individuals across roles. The most obvious choice, eLearning, can be kept up-to-date through a subscription model. Online courses must be engaging and relevant to achieve their intended purpose. Hot topics like mobile and cloud can attract more interest than wonky policy discussions. For developers, it is also possible to provide training directly through IDEs right at the time it’s needed. Remember that in some cases, building a new skill (such as code review) could be better suited for instructor-led training. Of course, training that sits around on the shelf does nobody any good.
[T2.5] Enhance satellite through training and events.
The SSG strengthens the social network by holding special events for the satellite. The satellite learns about advanced topics (e.g., the latest secure development techniques for building iOS applications) or hears from guest speakers.
A standing conference call with voluntary attendance does not address this activity, which is as much about building camaraderie as it is about sharing knowledge or organizational efficiency. There’s no substitute for face-to-face meetings, even if they happen only once or twice a year.

[T2.6] Include security resources in On-boarding.
The process for bringing new hires into the engineering organization requires a module about software security. The generic new hire process covers things like picking a good password and making sure people don’t tail you into the building, but this can be enhanced to cover topics such as secure coding, the SSDL, and internal security resources. The objective is to ensure that new hires enhance the security culture. Turnover in engineering organizations is generally high. Though a generic on-boarding module is useful, it does not take the place of a timely and more complete introductory software security course.

[T2.7] Identify satellite through training.
The satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or advanced knowledge of new tech stacks and development methodologies. Identifying this group is a step toward creating a social network that speeds the adoption of security into software development. One way to begin is to track the people who stand out during training courses. In general, a volunteer army may be easier to lead than one that is drafted.
T LEVEL 3

[T3.1] Reward progression through curriculum (certification or HR).

Knowledge is its own reward, but progression through the security curriculum brings other benefits too. Developers, testers, and others see a career advantage in learning about security. The reward system can be formal and lead to a certification or official mark in the HR system, or it can be less formal and use motivators such as praise letters for the satellite written at annual review time. Involving a corporate training department and/or HR can make security’s impact on career progression more obvious, but the SSG should continue to monitor security knowledge in the firm and not cede complete control or oversight.

[T3.2] Provide training for vendors or outsourced workers.

Spending time and effort helping suppliers get security right at the outset is easier than trying to figure out what they screwed up later on, especially if the agile team has sprinted on to other projects. In the best case, outsourced workers receive the same training given to employees. Training individual contractors is much more natural than training entire outsource firms and is a reasonable way to start. Of course, it’s important to train everyone who works on your software regardless of their employment status.

[T3.3] Host external software security events.

The organization highlights its security culture as a differentiator by hosting security events featuring external speakers and content. Good examples of this are Microsoft’s BlueHat and Intel’s Security Conference. Employees benefit from hearing outside perspectives, especially related to fast-moving technology areas. The organization, as a whole, benefits from putting its security credential(s) on display. Events open to just certain small groups will not result in the desired change.

[T3.4] Require an annual refresher.

Everyone involved in the SSDL is required to take an annual software security refresher course. The refresher keeps the staff up-to-date on security and ensures the organization doesn’t lose focus due to turnover, evolving methodologies, or changing deployment models. The SSG might use half a day to give an update on the security landscape and explain changes to policies and standards. A refresher can be rolled out as part of a firm-wide security day or in concert with an internal security conference.

[T3.5] Establish SSG office hours.

The SSG offers help to any and all comers during an advertised lab period or regularly scheduled office hours. By acting as an informal resource for people who want to solve security problems, the SSG leverages teachable moments and emphasizes the carrot over the stick. Office hours might be held one afternoon per week in the office of a senior SSG member. Roving office hours are also a possibility, with visits to particular product or application groups slated by request.
INTELLIGENCE: Attack Models (AM)

Attack Models capture information used to think like an attacker: threat modelling, abuse case development and refinement, data classification, and technology-specific attack patterns.

AM LEVEL 1

[AM1.2] Create a data classification scheme and inventory.
The organization agrees upon a data classification scheme and uses the scheme to inventory its software according to the kinds of data the software handles, regardless of whether the software is on or off premise. This allows applications to be prioritized by their data classification. Many classification schemes are possible—one approach is to focus on PII. Depending upon the scheme and the software involved, it could be easiest to first classify data repositories, then derive classifications for applications according to the repositories they use. Other approaches to the problem are possible. For example, data could be classified according to protection of intellectual property, impact of disclosure, exposure to attack, relevance to SOX, or geographic boundaries.

[AM1.3] Identify potential attackers.
The SSG identifies potential attackers in order to understand their motivations and capabilities. The outcome of this exercise could be a set of attacker profiles including generic sketches for categories of attackers and more detailed descriptions for noteworthy individuals. In some cases, a third-party vendor might be contracted to provide this information. Specific and contextual attacker information is almost always more useful than generic information copied from someone else’s list.

[AM1.5] Gather and use attack intelligence.
The SSG stays ahead of the curve by learning about new types of attacks and vulnerabilities. The information comes from attending conferences and workshops, monitoring attacker forums, and reading relevant publications, mailing lists, and blogs. Make Sun Tzu proud by knowing your enemy; engage with the security researchers who are likely to cause you trouble. In many cases, a subscription to a commercial service provides a reasonable way of gathering basic attack intelligence. Regardless of its origin, attack information must be made actionable and useful for software builders and testers.

AM LEVEL 2

[AM2.1] Build attack patterns and abuse cases tied to potential attackers.
The SSG prepares for security testing and architecture analysis by building attack patterns and abuse cases tied to potential attackers. These resources don’t have to be built from scratch for every application in order to be useful. Instead, there could be standard sets for applications with similar profiles. The SSG will add to the pile based on attack stories. For example, a story about an attack against poorly designed mobile application could lead to a mobile security attack pattern that drives a new type of testing. If a firm tracks fraud and monetary costs associated with particular attacks, this information can be used to guide the process of building attack patterns and abuse cases.

[AM2.2] Create technology-specific attack patterns.
The SSG creates technology-specific attack patterns to capture knowledge about attacks that target particular technologies. For example, if the organization’s cloud software relies on the cloud vendor’s security apparatus (e.g. Cryptography), the SSG could catalogue the quirks of the crypto package and how it might be exploited. Attack patterns directly related to the security frontier (e.g., mobile security and wearable computing) can be useful. Simply republishing general guidelines (e.g., “Ensure data are protected in transit”) and adding “for mobile applications” on the end does not constitute technology-specific attack patterns.
[AM2.5] Build and maintain a top N possible attacks list.
The SSG helps the organization understand attack basics by maintaining a living list of attacks most important to the firm and using it to drive change. This list combines input from multiple sources: observed attacks, hacker forums, industry trends, etc. The list does not need to be updated with great frequency and the attacks can be sorted in a coarse fashion. For example, the SSG might brainstorm twice per year to create lists of attacks the organization should be prepared to counter “now,” “soon,” and “someday.” In some cases, attack model information is used in a list-based approach to architecture analysis, helping to focus the analysis as in the case of STRIDE.

[AM2.6] Collect and publish attack stories.
To maximize the benefit from lessons that don’t always come cheap, the SSG collects and publishes stories about attacks against the organization. Over time, this collection helps the organization understand its history. Both successful and unsuccessful attacks can be noteworthy. Discussing historical information about software attacks has the effect of grounding software security in the reality of a firm. This is particularly useful in training classes to counter a generic approach over-focused on top 10 lists or irrelevant and outdated platform attacks. Hiding information about attacks from people building new systems does nothing to garner positive benefit from a negative happenstance.

[AM2.7] Build an internal forum to discuss attacks.
The organization has an internal forum where the SSG, the satellite, and others discuss attacks. The forum serves to communicate the attacker perspective. The SSG could maintain an internal mailing list where subscribers discuss the latest information on publicly known incidents. Dissection of attacks and exploits that are relevant to a firm are particularly helpful when they spur discussion of development mitigations. Simply republishing items from public mailing lists doesn’t achieve the same benefits as active discussion. Vigilance means never getting too comfortable.

AM LEVEL 3

[AM3.1] Have a science team that develops new attack methods.
The SSG has a science team that works to identify and defang new classes of attacks before real attackers even know they exist. Because the security implications of new technologies have not been fully explored in the wild, doing it yourself is sometimes the best way forward. This isn’t a penetration testing team finding new instances of known types of weaknesses—it’s a research group innovating new types of attacks. A science team may include well-known security researchers who publish their findings at conferences like Defcon.

[AM3.2: 2] Create and use automation to do what attackers will do.
The SSG arms testers and auditors with automation to do what attackers are going to do. For example, a new attack method identified by the science team could require a new tool. The SSG packages the new tool and distributes it to testers. The idea here is to push attack capability past what typical commercial tools and offerings encompass and then package that information for others to use. Tailoring these new tools to a firm’s particular technology stacks and potential attackers is a really good idea.
INTELLIGENCE: Security Features & Design (SFD)

The Security Features & Design practice is charged with creating usable security patterns for major security controls (meeting the standards defined in the Standards & Requirements practice), building middleware frameworks for those controls, and creating and publishing other proactive security guidance.

SFD LEVEL 1

[SFD1.1] Build and publish security features.

Some problems are best solved only once. Rather than have each project team implement all of their own security features (e.g., authentication, role management, key management, audit/log, cryptography, protocols), the SSG provides proactive guidance by building and publishing security features for other groups to use. Generic security features often have to be tailored for specific platforms, such as mobile. For example, a mobile crypto feature will need at least two versions to cover Android and iOS if it uses low-level system calls. Project teams benefit from implementations that come pre-approved by the SSG and the SSG benefits by not having to repeatedly track down the kinds of subtle errors that often creep into security features. The SSG can identify an implementation they like and promote it as the accepted solution.


Security is a regular part of the organization’s software architecture discussion. The architecture group takes responsibility for security the same way they take responsibility for performance, availability, or scalability. One way to keep security from falling out of the discussion is to have an SSG member attend regular architecture meetings. In other cases, enterprise architecture can help the SSG create secure designs that integrate properly into corporate design standards. Proactive engagement by the SSG is key to success. Moving a well-known system to the cloud means re-engaging the SSG. Assume nothing.

SFD LEVEL 2

[SFD2.1] Build secure-by-design middleware frameworks and common libraries.

The SSG takes a proactive role in software design by building or providing pointers to secure-by-design middleware frameworks or common libraries. In addition to teaching by example, this middleware aids architecture analysis and code review because the building blocks make it easier to spot errors. For example, the SSG could modify a popular web framework, such as Spring, to make it easy to meet input validation requirements. Eventually the SSG can tailor code review rules specifically for the components it offers (see [CR3.1 Use automated tools with tailored rules]).

When adopting a middleware framework (or any other widely used software), careful vetting for security before publication is important. Encouraging adoption and use of insecure middleware does not help the software security situation. Generic open source software security architectures, including OWASP ESAPI, should not be considered secure by design. Bolting security on at the end by calling a library is not the way to approach secure design.

[SFD2.2] Create SSG capability to solve difficult design problems.

When the SSG is involved early in the new project process, it contributes to new architecture and solves difficult design problems. The negative impact security has on other constraints (time to market, price, etc.) is minimized. If a skilled security architect from the SSG is involved in the design of a new protocol, he or she could analyse the security implications of existing protocols and identify elements that should be duplicated or avoided. Likewise, having a security architect understand the security implications of moving a seemingly well-understood application to the cloud saves lots of headaches later. Designing for security up front is more efficient than analyzing an existing design for security and then refactoring when flaws are uncovered. Some design problems will require specific expertise outside of the SSG.
SFD LEVEL 3
[SFD3.1] Form a review board or central committee to approve and maintain secure design patterns.
A review board or central committee formalizes the process for reaching consensus on design needs and security trade-offs. Unlike the architecture committee, this group is specifically focused on providing security guidance. The group also periodically reviews already-published design standards (especially around cryptography) to ensure that design decisions do not become stale or out of date. A review board can also reign in the chaos often associated with the adoption of new technologies where development groups might otherwise head out into the jungle on their own without ever engaging the SSG.

[SFD3.2] Require use of approved security features and frameworks.
Implementers must take their security features and frameworks from an approved list. There are two benefits:
1) developers do not spend time re-inventing existing capabilities; and,
2) review teams do not have to contend with finding the same old defects in brand new projects or when new platforms are adopted.
In particular, the more a project uses proven components, the easier architecture analysis and code review become.
Re-use is a major advantage of consistent software architecture and is particularly helpful for agile development.

[SFD3.3] Find and publish mature design patterns from the organization.
The SSG fosters centralized design reuse by collecting design patterns from across the organization and publishing them for everyone to use. A section of the SSG website could promote positive elements identified during architecture analysis so that good ideas are spread. This process should be formalized. An ad hoc, accidental noticing is not sufficient. In some cases, a central architecture or technology team facilitates and enhances this activity. Common design patterns make building software faster.

INTELLIGENCE: Standards & Requirements (SR)
The Standards & Requirements practice involves eliciting explicit security requirements from the organization, determining which COTS to recommend, building standards for major security controls (such as authentication, input validation, and so on), creating security standards for technologies in use, and creating a standards review board.

SR LEVEL 1
[SR1.1] Create security standards.
Software security requires much more than security features, but security features are part of the job as well. The SSG meets the organization’s demand for security guidance by creating standards that explain the accepted way to adhere to policy and carry out specific security-centric operations. A standard might describe how to perform authentication on an Android device or how to determine the authenticity of a software update. Standards can be deployed in a variety of ways. In some cases, standards and guidelines can be automated in development environments (e.g., worked into an IDE). In other cases, guidance can be explicitly linked to code examples or even containers to make them more actionable and relevant. Standards that are not widely adopted and enforced are not really standards.

[SR1.2] Create a security portal.
The organization has a well-known central location for information about software security. Typically, this is an internal website maintained by the SSG. People refer to the site for the latest and greatest on security standards and requirements as well as other resources provided by the SSG. An interactive wiki is better than a
static portal with guideline documents that rarely change. Organizations can supplement these materials with mailing lists and face-to-face meetings.

[SR1.3] Translate compliance constraints to requirements.
Compliance constraints are translated into software requirements for individual projects. This is a linchpin in the organization’s compliance strategy—by representing compliance constraints explicitly with requirements, demonstrating compliance becomes a manageable task. For example, if the organization routinely builds software that processes credit card transactions, PCI-DSS compliance could play a role in the SSDL during the requirements phase. In other cases, technology standards built for international interoperability reasons can include security guidance. Representing these standards as requirements helps with traceability and visibility in the case of audit.
Bonus points for codifying the requirements in re-usable code or containers.

SR LEVEL 2

[SR2.2] Create a standards review board.
The organization creates a standards review board to formalize the process used to develop standards and ensure that all stakeholders have a chance to weigh in. The review board could operate by appointing a champion for any proposed standard. The onus is on the champion to demonstrate that the standard meets its goals and to get approval and buy-in from the review board. Enterprise architecture or enterprise risk groups sometimes take on the responsibility of creating and managing standards review boards.

[SR2.3] Create standards for technology stacks.
The organization standardizes on specific technology stacks. For the SSG, this means a reduced workload because the group does not have to explore new technology risks for every new project. Ideally, the organization will create a secure base configuration for each technology stack, further reducing the amount of work required to use the stack safely. A stack might include an operating system, a database, an application server, and a runtime environment for a managed language. The security frontier is a good place to find traction. Currently, mobile technology stacks and platforms as well as cloud-based technology stacks are two areas where specific attention to security pays off.

[SR2.4] Identify open source.
The first step toward managing risk introduced by open source is to identify the open source components in use across the portfolio and really understand which dependencies they drag on stage. It’s not uncommon to discover old versions of components with known vulnerabilities or multiple versions of the same component. Automated tools for finding open source, whether whole components or large chunks of borrowed code, are one way to approach this activity. An informal annual review or a process that relies solely on developers asking for permission does not generate satisfactory results. At the next level of maturity, this activity is subsumed by a policy constraining the use of open source.

[SR2.5] Create SLA boilerplate.
The SSG works with the legal department to create a standard SLA boilerplate that is used in contracts with vendors and outsource providers to require software security efforts. This includes cloud providers. The legal department understands that the boilerplate also helps prevent compliance and privacy problems. Under the agreement, vendors and outsource providers must meet company software security standards. Boilerplate language may call out software security vendor control solutions such as BSIMM measurements or BSIMM scores.

[SR2.6] Use secure coding standards.
Secure coding standards help developers avoid the most obvious bugs and provide ground rules for code review.
Secure coding standards are necessarily specific to a programming language or platform and can address the use of popular frameworks and libraries. Mobile platforms need their own specific coding standards. If the organization already has coding standards for other purposes, the secure coding standards should build upon them. A clear set of secure coding standards is a good way to guide both manual and automated code review, as well as beefing up security training with relevant examples. Remember, guidance does not a standard make.

**SR LEVEL 3**

**[SR3.1] Control open source risk.**

The organization has control over its exposure to the vulnerabilities that come along with using open source components and their army of dependencies. Use of open source could be restricted to pre-defined projects. It could also be restricted to open source versions that have been through an SSG security screening process, had unacceptable vulnerabilities remediated, and are made available only through internal repositories. Legal often spearheads additional open source controls due to the “viral” license problem associated with GPL code. Getting legal to understand security risks can help move an organization to practice decent open source hygiene. Of course, this control must be applied across the software portfolio.

**[SR3.2] Communicate standards to vendors.**

The SSG works with vendors to educate them and promote the organization’s security standards. A healthy relationship with a vendor cannot be guaranteed through contract language alone. The SSG engages with vendors, discusses the vendor’s security practices and explains in concrete terms (rather than legalese) what the organization expects of the vendor. Any time a vendor adopts the organization’s security standards, it’s a clear win. When a firm’s SSDL is available publicly, communication regarding software security expectations is easier. Likewise, sharing internal practices and measures can make expectations very clear.
SSDL TOUCHPOINTS: Architecture Analysis (AA)

Architecture Analysis encompasses capturing software architecture in concise diagrams, applying lists of risks and threats, adopting a process for review (such as STRIDE or Architecture Risk Analysis), and building an assessment and remediation plan for the organization.

AA LEVEL 1

[AA1.1] Perform security feature review.
To get started in architecture analysis, center the process on a review of security features. Security-aware reviewers first identify the security features in an application (authentication, access control, use of cryptography, etc.) then study the design looking for problems that would cause these features to fail at their purpose or otherwise prove insufficient.

For example, a system that was subject to escalation of privilege attacks because of broken access control or a mobile application that stashed away PII on local storage would both be identified in this kind of review. At higher levels of maturity, the activity of reviewing features is eclipsed by a more thorough approach to architecture analysis. In some cases, use of the firm’s secure-by-design components can streamline this process.

[AA1.2] Perform design review for high-risk applications.
The organization learns about the benefits of architecture analysis by seeing real results for a few high-risk, high-profile applications. The reviewers must have some experience performing detailed design review and breaking the architecture being considered, especially for new platforms or environments. In all cases, design review produces a set of architecture flaws and a plan to mitigate them. If the SSG is not yet equipped to perform an in-depth architecture analysis, it uses consultants to do this work. Ad hoc review paradigms that rely heavily on expertise can be used here, though in the long run they do not scale. A review focused only on whether a software project has performed the right process steps will not generate expected results.

[AA1.3] Have SSG lead design review efforts.
The SSG takes a lead role in architecture analysis by performing design review to build the organization’s ability to uncover design flaws. Breaking an architecture is enough of an art that the SSG must be proficient at it before they can turn the job over to the architects, and proficiency requires practice. The SSG cannot be successful on its own either—it’s likely they’ll need help from architects or implementers to understand the design. With a clear design in hand, the SSG might carry out the detailed review with a minimum of interaction with the project team. At higher levels of maturity, the responsibility for leading review efforts shifts towards software architects. Approaches to architecture analysis (and threat modelling) evolve over time. Do not expect to set a process and use it forever.

[AA1.4] Use a risk questionnaire to rank applications.
To facilitate security feature and design review processes, the SSG uses a risk questionnaire to collect basic information about each application so that it can determine a risk classification and prioritization scheme.

Questions might include, “Which programming languages is the application written in?”, “Who uses the application?” and “Does the application have a mobile client?” A qualified member of the application team completes the questionnaire. The questionnaire is short enough to be completed in a matter of hours. The SSG might use the answers to bucket the application as high, medium, or low risk.

Because a risk questionnaire can be easy to game, it’s important to put some spot checking for validity and accuracy into place. An overreliance on self-reporting or automation can render this activity impotent.

AA LEVEL 2

[AA2.1] Define and use AA process.
The SSG defines and documents a process for architecture analysis and applies it in the design reviews it conducts.
The process includes a standardized approach for thinking about attacks and security properties, and the associated risk. The process is defined rigorously enough that people outside the SSG can be taught to carry it out.
Particular attention should be paid to documentation of both the architecture under review and any security flaws uncovered. Tribal knowledge doesn’t count as a defined process. Microsoft’s STRIDE and Cigital’s ARA are examples of this process. Note that even these two methodologies for architecture analysis have evolved greatly over time.
Make sure to access up-to-date sources for architecture analysis information because many early publications are outdated and no longer apply.

[AA2.2] Standardize architectural descriptions (including data flow).
AA processes use an agreed-upon format for describing architecture, including a means for representing data flow. This format, combined with an architecture analysis process, makes architecture analysis tractable for people who are not security experts. In the case of cloud applications, data is likely to flow across the Internet. A network diagram is useful in this case, but the description should go into detail about how the software itself is structured.

A standard architecture description can be enhanced to provide an explicit picture of information assets that require protection. Standardized icons that are consistently used in UML diagrams, Visio templates, and whiteboard squiggles are especially useful.

[AA2.3] Make the SSG available as AA resource or mentor.
To build an architecture analysis capability outside of the SSG, the SSG advertises itself as a resource or mentor for teams who ask for help using the AA process to conduct their own design review and proactively seek projects to get involved with. The SSG will answer architecture analysis questions during office hours, and in some cases, might assign someone to sit side-by-side with the architect for the duration of the analysis. In the case of high-risk applications or products, the SSG plays a more active mentorship role in applying the AA process.

AA LEVEL 3

[AA3.1] Have software architects lead design review efforts.
Software architects throughout the organization lead the architecture analysis process most of the time. The SSG still might contribute to architecture analysis in an advisory capacity or under special circumstances. This activity requires a well-understood and well-documented architecture analysis process. Even in that case, consistency is very difficult to attain because breaking architectures requires so much experience.

[AA3.2] Drive analysis results into standard architecture patterns.
Failures identified during architecture analysis are fed back to the security design committee so that similar mistakes can be prevented in the future through improved design patterns (see [SFD3.1 Form a review board or central committee to approve and maintain secure design patterns]). Security design patterns can interact in surprising ways that break security. The architecture analysis process should be applied even when vetted design patterns are in standard use.
SSDL TOUCHPOINTS: Code Review (CR)

The Code Review practice includes use of code review tools, development of tailored rules, customized profiles for tool use by different roles (for example, developers versus auditors), manual analysis, and tracking/measuring results.

CR LEVEL 1

[CR1.2] Have SSG perform ad hoc review.
The SSG performs an ad hoc code review for high-risk applications in an opportunistic fashion. For example, the SSG might follow up the design review for high-risk applications with a code review. At higher maturity levels, replace ad-hoc targeting with a systematic approach. SSG review could involve the use of specific tools and services, or it might be manual. When new technologies pop up, new approaches to code review might become necessary. An ad-hoc exploration is okay.

[CR1.4] Use automated tools along with manual review.
Incorporate static analysis into the code review process to make code review more efficient and more consistent.
The automation doesn’t replace human judgment, but it does bring definition to the review process and security expertise to reviewers who are not security experts. Note that a specific tool might not cover your entire portfolio, especially when new languages are involved, but that’s no excuse not to review your code. A firm may use an external service vendor as part of a formal code review process for software security. This service should be explicitly connected to a larger SSDL applied during software development and not just “check the security box” on the path to deployment.

[CR1.5] Make code review mandatory for all projects.
Code review is a mandatory release gate for all projects under the SSG’s purview. Lack of code review or unacceptable results will stop the release train. While all projects must undergo code review, the review process might be different for different kinds of projects. The review for low-risk projects might rely more heavily on automation and the review for high-risk projects might have no upper bound on the amount of time spent by reviewers. In most cases, a code review gate with a minimum acceptable standard forces projects that don’t pass to be fixed and re-evaluated before they ship.

[CR1.6] Use centralized reporting to close the knowledge loop and drive training.
The bugs found during code review are tracked in a centralized repository.
This repository makes it possible to do summary reporting and trend reporting for the organization. The SSG can use the reports to demonstrate progress and drive the training curriculum (see [SM2.5 Identify metrics and use them drive budgets]).
Code review information can be incorporated into a CISO-level dashboard that includes feeds from other parts of the security organization. Likewise, code review information can be fed into a development-wide project tracking system that rolls up several diverse software security feeds (for example, penetration tests, security testing, black box testing, white box testing, etc.).
Don’t forget that individual bugs make excellent training examples.
CR LEVEL 2

[CR2.5] Assign tool mentors.
Mentors are available to show developers how to get the most out of code review tools. If the SSG is most skilled with the tools, it could use office hours to help developers establish the right configuration or get started interpreting results. Alternatively, someone from the SSG might work with a development team for the duration of the first review they perform. Centralized use of a tool can be distributed into the development organization over time through the use of tool mentors.

[CR2.6] Use automated tools with tailored rules.
Customize static analysis to improve efficiency and reduce false positives. Use custom rules to find errors specific to the organization’s coding standards or custom middleware. Turn off checks that aren’t relevant. The same group that provides tool mentoring will likely spearhead the customization. Tailored rules can be explicitly tied to proper usage of technology stacks in a positive sense and avoidance of errors commonly encountered in a firm’s code base in a negative sense.

[CR2.7] Use a top N bugs list (real data preferred).
The SSG maintains a list of the most important kinds of bugs that it wants to eliminate from the organization’s code and uses it to drive change. The list helps focus the organization’s attention on the bugs that matter most.

It’s okay to start with a generic list pulled from public sources, but a list is much more valuable if it’s specific to the organization and built from real data gathered from code review, testing, and actual incidents. The SSG can periodically update the list and publish a “most wanted” report.

Some firms use multiple tools and real code base data to build Top N lists, not constraining themselves to a particular service or tool. One potential pitfall with a Top N list is the problem of looking for your keys only under the street light.” For example, the OWASP Top 10 list rarely reflects an organization’s bug priorities. Simply sorting the day’s bug data by number of occurrences doesn’t produce a satisfactory Top N list because these data change so often.
CR LEVEL 3

[CR3.2] Build a factory.
Combine assessment results so that multiple analysis techniques feed into one reporting and remediation process.

The SSG might write scripts to invoke multiple detection techniques automatically and combine the results into a format that can be consumed by a single downstream review and reporting solution. Analysis engines may combine static and dynamic analysis. Different review streams, such as mobile versus standard approaches, can be unified with a factory. The tricky part of this activity is normalizing vulnerability information from disparate sources that use conflicting terminology. In some cases, a CWE-like approach can help with nomenclature. Combining multiple sources helps drive better informed risk mitigation decisions.

[CR3.3] Build a capability for eradicating specific bugs from the entire codebase.
When a new kind of bug is found, the SSG writes rules to find it and uses the rules to identify all occurrences of the new bug throughout the entire codebase. It is possible to eradicate the bug type entirely without waiting for every project to reach the code review portion of its lifecycle. A firm with only a handful of software applications will have an easier time with this activity than firms with a very large number of large applications.

Automated code review is used to identify dangerous code written by malicious in-house developers or outsource providers. Examples of malicious code that could be targeted include backdoors, logic bombs, time bombs, nefarious communication channels, obfuscated program logic, and dynamic code injection. Although out-of-the-box automation might identify some generic malicious-looking constructs, custom rules for static analysis tools used to codify acceptable and unacceptable code patterns in the organization’s codebase will quickly become a necessity. Manual code review for malicious code is a good start but is insufficient to complete this activity.

A violation of the organization’s secure coding standards is sufficient grounds for rejecting a piece of code. Code review is objective—it shouldn’t devolve into a debate about whether or not bad code is exploitable. The enforced portion of the standard could start out being as simple as a list of banned functions. In some cases, coding standards for developers are published specific to technology stacks (for example, guidelines for C++, Spring, or Swift) and then enforced during the code review process or directly in the IDE. Standards can be positive (“do it this way”) as well as negative (“do not use this API”).
SSDL TOUCHPOINTS: Security Testing (ST)

The Security Testing practice is concerned with pre-release testing, including integrating security into standard quality assurance processes. The practice includes use of black box security tools (including fuzz testing) as a smoke test in QA, risk-driven white box testing, application of the attack model, and code coverage analysis. Security testing focuses on vulnerabilities in construction.

ST LEVEL 1

[ST1.1] Ensure QA supports edge/boundary value condition testing.
The QA team goes beyond functional testing to perform basic adversarial tests. They probe simple edge cases and boundary conditions. No attacker skills required. When QA understands the value of pushing past standard functional testing using acceptable input, they begin to move slowly toward “thinking like a bad guy.” A discussion of boundary value testing leads naturally to the notion of an attacker probing the edges on purpose. What happens when you enter the wrong password over and over?

[ST1.3] Drive tests with security requirements and security features.
Testers target declarative security mechanisms derived from requirements and security features. For example, a tester could try to access administrative functionality as an unprivileged user or verify that a user account becomes locked after some number of failed authentication attempts. For the most part, security features can be tested in a similar fashion to other software features. Security mechanisms based on requirements such as account lockout, transaction limitations, entitlements, and so on are also tested. Of course, software security is not security software, budgeting started with features is easy. New deployment models, such as cloud, may require novel test approaches.

ST LEVEL 2

[ST2.1] Integrate black box security tools into the QA process.
The organization uses one or more black box security testing tools as part of the quality assurance process. The tools are valuable because they encapsulate an attacker’s perspective, albeit in a generic fashion. Tools such as IBM Security AppScan or HP WebInspect are relevant for web applications, and fuzzing frameworks such as Codenomicon are applicable for most network protocols. In some situations, other groups might collaborate with the SSG to apply the tools. For example, a testing team could run the tool, but come to the SSG for help interpreting the results. Because of the way testing is integrated into agile development approaches, black box tools might be used directly by the agile team. Regardless of who runs the black box tool, the testing should be properly integrated into the QA cycle of the SSDL.

[ST2.4] Share security results with QA.
The SSG routinely shares results from security reviews with the QA department. Agile development methodologies make this easier because of the way testing is integrated in a cross-functional team. Over time, QA engineers learn the security mind-set. Using security results to inform and evolve particular testing patterns can be a powerful mechanism leading to better security testing. This activity benefits from an engineering-focused QA function that is highly technical.

[ST2.5] Include security tests in QA automation.
Security tests run alongside functional tests as part of automated regression testing. The same automation framework houses both. Security testing is part of the routine. Security tests can be driven from abuse cases identified earlier in the lifecycle or tests derived from creative tweaks of functional tests.
[ST2.6] Perform fuzz testing customized to application APIs.
Test automation engineers or agile team members customize a fuzzing framework to the organization’s APIs. They could begin from scratch or use an existing fuzzing toolkit, but customization goes beyond creating custom protocol descriptions or file format templates. The fuzzing framework has a built-in understanding of the application interfaces it calls into. Test harnesses developed explicitly for particular applications can make good places to integrate fuzz testing.

ST LEVEL 3

[ST3.3] Drive tests with risk analysis results.
Testers use architecture analysis results to direct their work. For example, if architecture analysis concludes, “the security of the system hinges on the transactions being atomic and not being interrupted partway through,” then torn transactions will be become a primary target in adversarial testing. Adversarial tests like these can be developed according to risk profile—high-risk flaws first.

[ST3.4] Leverage coverage analysis.
Testers measure the code coverage of their security tests to identify code that isn’t being exercised. Code coverage drives increased security testing depth. Standard-issue black box testing tools achieve exceptionally low coverage, leaving a majority of the software under test unexplored. Don’t let this happen to your tests. Using standard measurements for coverage such as function coverage, line coverage, or multiple condition coverage is fine.

[ST3.5] Begin to build and apply adversarial security tests (abuse cases).
Testing begins to incorporate test cases based on abuse cases. Testers move beyond verifying functionality and take on the attacker’s perspective. For example, testers might systematically attempt to replicate incidents from the organization’s history. Abuse and misuse cases based on the attacker’s perspective can also be driven from security policies, attack intelligence and guidelines. This turns the corner from testing features to attempting to break the software under test.
DEPLOYMENT: Penetration Testing (PT)
The Penetration Testing practice involves standard outside-in testing of the sort carried out by security specialists. Penetration testing focuses on vulnerabilities in the final configuration and provides direct feeds to defect management and mitigation.

PT LEVEL 1
[PT1.1] Use external penetration testers to find problems.
Many organizations aren’t willing to address software security until there’s unmistakable evidence that the organization isn’t somehow magically immune to the problem. If security has not been a priority, external penetration testers can demonstrate that the organization’s code needs help. Penetration testers could be brought in to break a high-profile application to make the point. Over time, the focus of penetration testing moves from “I told you our stuff was broken” to a smoke test and sanity check done before shipping. External penetration testers bring a new set of eyes to the problem.

[PT1.2] Feed results to the defect management and mitigation system.
Penetration testing results are fed back to development through established defect management or mitigation channels, and development responds using their defect management and release process. The exercise demonstrates the organization’s ability to improve the state of security. Many firms are beginning to emphasize the critical importance of not just identifying but, more importantly, fixing security problems. One way to ensure attention is to add a security flag to the bug tracking and defect management system. Evolving DevOps and integrated team structures do not eliminate the need for formalized defect management systems.

[PT 1.3] Use penetration testing tools internally.
The organization creates an internal penetration testing capability that uses tools. This capability can be part of the SSG or part of a specialized and trained team elsewhere in the organization. The tools improve efficiency and repeatability of the testing process and are often necessary in agile environments. Tools can include off-the-shelf products, standard issue network penetration tools that understand the application layer, and hand-written scripts.
PT LEVEL 2
[PT2.2] Provide penetration testers with all available information.
Penetration testers, whether internal or external, use all available information about their target. Penetration testers can do deeper analysis and find more interesting problems when they have source code, design documents, architecture analysis results, and code review results. Give penetration testers everything you have created throughout the SSDL and ensure they use it. If your penetration tester doesn’t ask for the code, you need a new penetration tester.

[PT2.3] Schedule periodic penetration tests for application coverage.
Periodically test all applications in the SSG’s purview according to an established schedule, which could be tied to the calendar or to the release cycle. The testing serves as a sanity check and helps ensure yesterday’s software isn’t vulnerable to today’s attacks. This also helps maintain the security of software configurations and environments, especially in the cloud. High-profile applications might get a penetration test at least once a year. One important aspect of periodic testing is to make sure that the problems identified in a penetration test are actually fixed and they don’t creep back into the build.

PT LEVEL 3
[PT3.1] Use external penetration testers to perform deep-dive analysis.
The organization uses external penetration testers to do deep-dive analysis for critical projects and to introduce fresh thinking into the SSG. These testers are experts and specialists; they keep the organization up to speed with the latest version of the attacker’s perspective and have a track record for breaking the type of software being tested. Skilled penetration testers will always break a system. The question is whether they demonstrate new kinds of thinking about attacks that can be useful when designing, implementing, and hardening new systems. Creating new types of attacks from threat intelligence and abuse cases prevents checklist-driven approaches that only look for known types of problems and is pretty much essential when it comes to new technology.

[PT3.2] Have the SSG customize penetration testing tools and scripts.
The SSG either creates penetration testing tools or adapts publicly-available tools so they can more efficiently and comprehensively attack the organization’s systems. Tools improve the efficiency of the penetration testing process without sacrificing the depth of problems the SSG can identify. Automation can be particularly valuable under agile methodologies because it helps teams go faster. Tools that can be tailored are always preferable to generic tools.
This activity considers both the depth of tests and their scope.
DEPLOYMENT: Software Environment (SE)
The Software Environment practice concerns itself with OS and platform patching, web application firewalls, installation and configuration documentation, application monitoring, change management and, ultimately, code signing.

SE LEVEL 1
[SE1.1] Use application input monitoring.
The organization monitors the input to software it runs in order to spot attacks. For web code, a web application firewall (WAF) can do the job. The SSG could be responsible for the care and feeding of the system. Incident response is not part of this activity. Defanged WAFs that write log files can be useful if somebody reviews the logs periodically. On the other hand, a WAF that’s unmonitored makes no noise when an application falls in the woods.

[SE1.2] Ensure host and network security basics are in place.
The organization provides a solid foundation for software by ensuring that host and network security basics are in place. It is common for operations security teams to be responsible for duties such as patching operating systems, maintaining firewalls, and properly configuring cloud services. Doing software security before network security is like putting on your pants before putting on your underwear.

SE LEVEL 2
[SE2.2] Publish installation guides.
The SSDL requires the creation of an installation guide or a clearly-described configuration, such as a container, to help deployment teams and operators install and configure the software securely. If special steps are required to ensure a deployment is secure, the steps are either outlined in the installation guide or explicitly noted in deployment automation. The guide should include discussion of COTS components. In some cases, installation guides are distributed to customers who buy the software. Make sure that all deployment automation can be understood by smart humans and not just a machine. Evolving DevOps and integrated team structures do not eliminate the need for human-readable guidance. Of course, secure by default is always the best way to go.

[SE2.4] Use code signing.
The organization uses code signing for software published across trust boundaries. Code signing is particularly useful for protecting the integrity of software that leaves the organization’s control, such as shrink-wrapped applications or thick clients. The fact that some mobile platforms require application code to be signed does not indicate institutional use of code signing.
SE LEVEL 3

[SE3.2] Use code protection.
To protect intellectual property and make exploit development harder, the organization erects barriers to reverse engineering. This is particularly important for widely-distributed mobile applications. Obfuscation techniques could be applied as part of the production build and release process. Employing platform-specific controls such as Data Execution Prevention (DEP), Safe Structured Error Handling (SafeSEH), and Address Space Layout Randomization (ASLR) can make exploit development more difficult.

[SE3.3] Use application behavior monitoring and diagnostics.
The organization monitors the behavior of production software looking for misbehavior and signs of attack. This activity goes beyond host and network monitoring to look for problems that are specific to the software, such as indications of fraud. Intrusion detection and anomaly detection systems at the application level may focus on an application’s interaction with the operating system (through system calls) or with the kinds of data that an application consumes, originates, and manipulates.

[SE3.4] Use application containers.
The organization uses application containers to support its software security goals. The primary drivers for using containers include ease of deployment, a tighter coupling of applications with their dependencies, and isolation without the overhead of deploying a full OS on a virtual machine. Containers provide a convenient place for security controls to be applied and updated consistently. Containers used in development or test environments without reference to security do not count.
DEPLOYMENT: Configuration Management & Vulnerability Management (CMVM)

The Configuration Management & Vulnerability Management practice concerns itself with patching and updating applications, version control, defect tracking and remediation, and incident handling.

CMVM LEVEL 1

[CMVM1.1] Create or interface with incident response.

The SSG is prepared to respond to an incident and is regularly included in the incident response process. The group either creates its own incident response capability or regularly interfaces with the organization’s existing incident response team. A regular meeting between the SSG and the incident response team can keep information flowing in both directions. Sometimes cloud service providers need to be looped in. In many cases, software security initiatives have evolved from incident response teams who began to realize that software vulnerabilities were the bane of their existence.

[CMVM1.2] Identify software defects found in operations monitoring and feed them back to development.

Defects identified through operations monitoring are fed back to development and used to change developer behavior. The contents of production logs can be revealing (or can reveal the need for improved logging). In some cases, providing a way to enter incident triage data into an existing bug tracking system (many times making use of a special security flag) seems to work. The idea is to close the information loop and make sure security problems get fixed. In the best of cases, processes in the SSDL can be improved based on operational data.
CMVM LEVEL 2
[CMVM2.1] Have emergency codebase response.
The organization can make quick code changes when an application is under attack. A rapid-response team works in conjunction with the application owners and the SSG to study the code and the attack, find a resolution, and push a patch into production. Often, the emergency response team is the development team itself, especially when agile methodologies are in use. Fire drills don’t count; a well-defined process is required. A process that has never been used may not actually work.

[CMVM2.2] Track software bugs found in operations through the fix process.
Defects found in operations are fed back to development, entered into established defect management systems and tracked through the fix process. This capability could come in the form of a two-way bridge between the bug finders and the bug fixers. Make sure the loop is closed completely. Setting a security flag in the bug-tracking system can help facilitate tracking.

[CMVM2.3] Develop an operations inventory of applications.
The organization has a map of its software deployments. If a piece of code needs to be changed, operations or DevOps can reliably identify all of the places where the change needs to be installed. Sometimes common components shared between multiple projects are noted so that when an error occurs in one application, other applications that share the same components can be fixed as well.
CMVM LEVEL 3

[CMVM3.1] Fix all occurrences of software bugs found in operations.
The organization fixes all instances of each software bug found during operations and not just the small number of instances that have triggered bug reports. This requires the ability to re-examine the entire codebase when new kinds of bugs come to light. One way to approach this is to create a rule set that generalizes a deployed bug into something that can be scanned for using automated code review.

[CMVM3.2] Enhance the SSDL to prevent software bugs found in operations.
Experience from operations leads to changes in the SSDL. The SSDL is strengthened to prevent the reintroduction of bugs found during operations. To make this process systematic, each incident response post mortem could include a “feedback to SSDL” step. This works best when root cause analysis pinpoints where in the SDLC an error could have been introduced or slipped by uncaught. Cross-functional agile teams might have an easier time with this because all the players are involved. An ad hoc approach is not sufficient.

[CMVM3.3] Simulate software crisis.
The SSG simulates high-impact software security crises to ensure software incident response capabilities minimize damage. Simulations could test for the ability to identify and mitigate specific threats or, in other cases, could begin with the assumption that a critical system or service is already compromised and evaluate the organization’s ability to respond. When simulations model successful attacks, an important question to consider is the time period required to clean up. Regardless, simulations must focus on security-relevant software failure and not on natural disasters or other types of emergency response drills. If the data center is burning to the ground, the SSG won’t be among the first responders.

[CMVM3.4] Operate a bug bounty program.
The organization solicits vulnerability reports from external researchers and pays a bounty for each verified and accepted vulnerability received. Pay-outs typically follow a sliding scale linked to multiple factors, such as vulnerability type (e.g., remote code execution is worth $10,000 versus CSRF is worth $750), exploitability (demonstrable exploits command much higher pay-outs), or specific services and software versions (widely-deployed or critical services warrant higher pay-outs). Ad hoc or short-duration activities, such as capture-the-flag contests, do not count.