www.pwc.com

Secure Development Lifecycles: Motivation & Overview

SecAppDev 2019

Bart De Win

pwc

Bart De Win ?

- •20 years of Information Security Experience
  - •Ph.D. in Computer Science - Application Security
- •Author of >60 scientific publications
- •ISC² CSSLP & CISSP certified
- •Director @ Cyber&Privacy PwC Belgium:
  - •Leading the Threat & Vuln. Mngt. team
  - •(Web) Application tester (arch. review, code review, dynamic review, ...)
  - •Proficiency in Secure Software Development Lifecycle (SDLC) and Software Quality (ISO25010)
- OWASP SAMM co-leader
- Contact me at bart.de.win@pwc.com

Agenda

1. **Setting the Scene**
2. Process Models
3. Modern Development
4. Maturity Models
5. Good Practices
6. Conclusion

What's in a name …

“Secure Development”

Application Security Problem

SPEED IS THE KEY TO SUCCESS.
IS IT OKAY TO DO THINGS WRONG IF WE'RE REALLY, REALLY FAST?
UM... NO.
NOW I'M ALL CONFUSED. THANK YOU VERY MUCH.
www.dilbert.com
scottadams@aol.com
© 2000 United Feature Syndicate, Inc.

Copyright © 2000 United Feature Syndicate, Inc.
Redistribution in whole or in part prohibited

Technology stacks

Multi-channel

Adaptability

Software complexity

Outsourcing

Api's

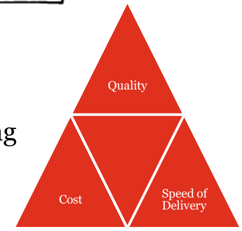
Integration

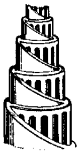
Faster

Cloud

Usability

Open Source





Secure Development Lifecycles: Motivation and Overview

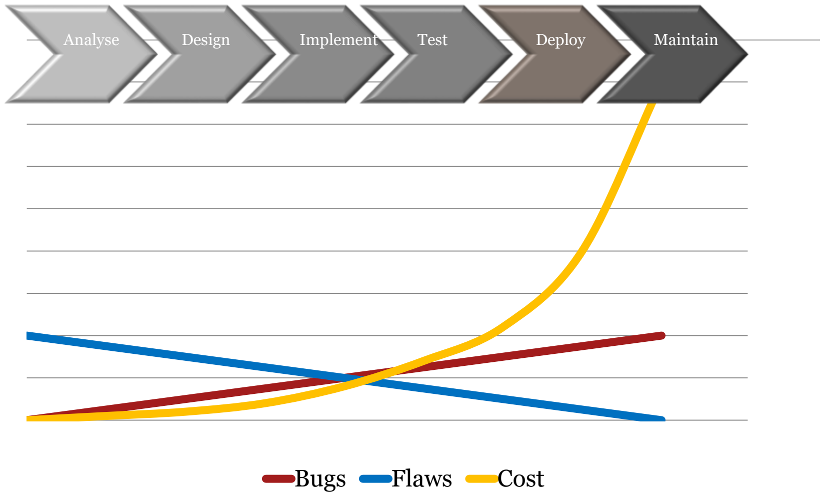
SecAppDev 2019
5

Your view on your software?

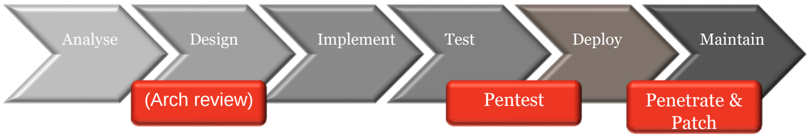
Secure Development Lifecycles: Motivation and Overview

SecAppDev 2019
6

Application Security during Software Development



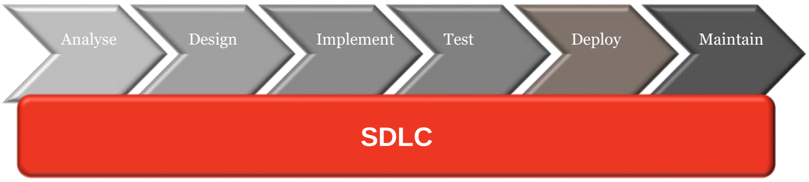
The “classic” approach to secure software



Problematic, since:

- Focus on bugs, not flaws
- Penetration can cause major harm
- Not cost efficient
- No security assurance
  - All bugs found ?
  - Bug fix fixes all occurences ? (also future ?)
  - Bug fix might introduce new security vulnerabilities

Secure Development Lifecycle ?

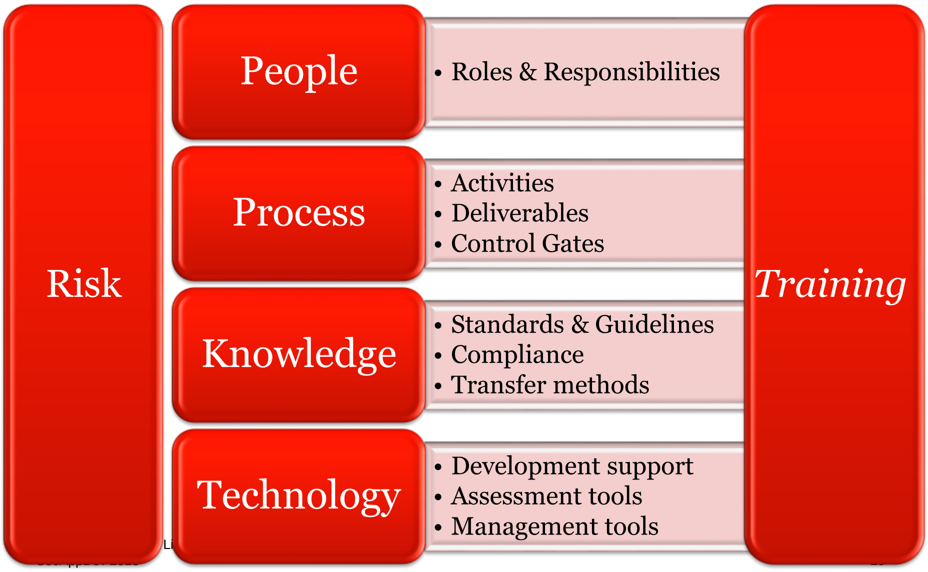


Enterprise-wide software security program

- Strategic approach to assure software quality
- Goal is to increase systematicity and avoid surprises
- Goal is NOT to have fully secure applications
- Focus on security *functionality* and security *hygiene*

Shift Left

SDLC Cornerstones

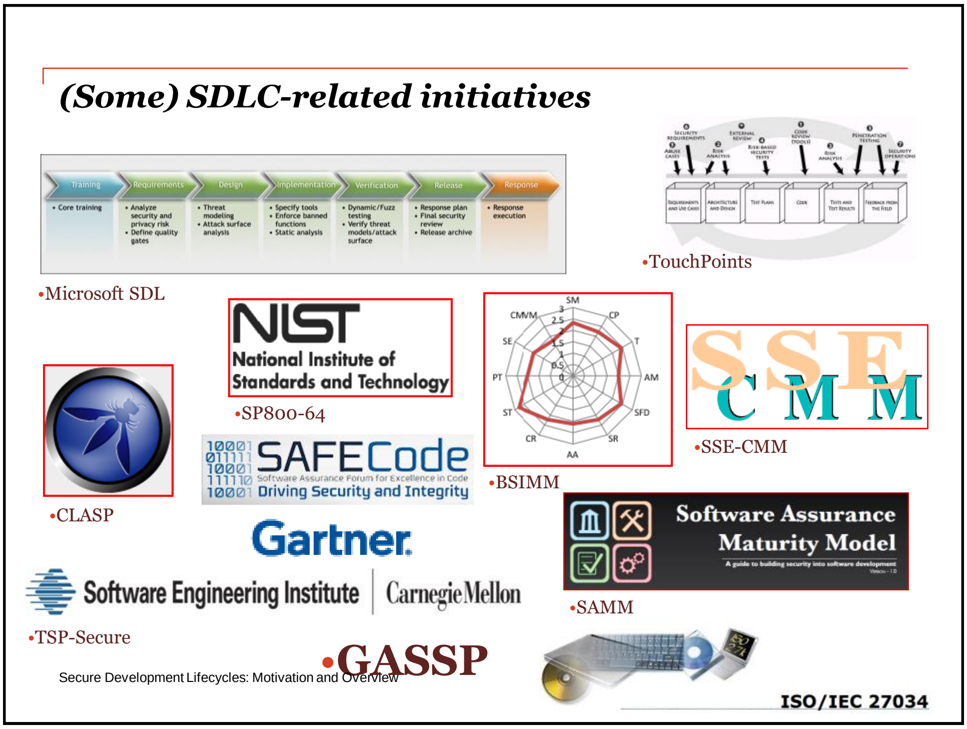


- Roles & Responsibilities

- Activities
- Deliverables
- Control Gates

- Standards & Guidelines
- Compliance
- Transfer methods

- Development support
- Assessment tools
- Management tools



(Some) SDLC-related initiatives

- Microsoft SDL

- TouchPoints

- SP800-64

- BSIMM

- SSE-CMM
- CLASP

SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity
Gartner
Software Engineering Institute | Carnegie Mellon

- TSP-Secure

Software Assurance Maturity Model
A guide to building security into software development

- SAMM
- GASSP

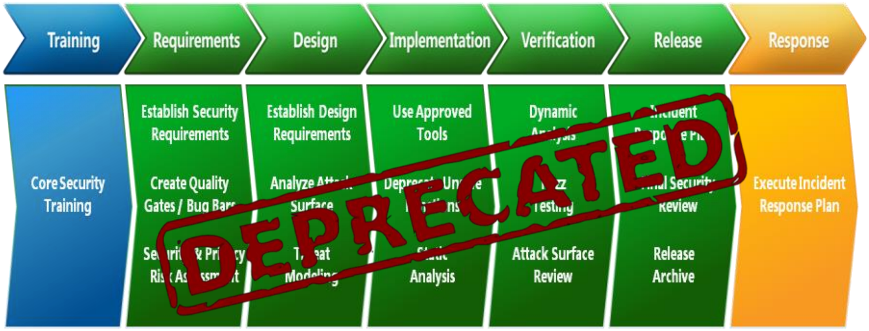
Secure Development Lifecycles: Motivation and Overview

Agenda

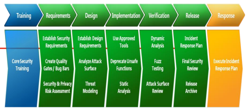
1. Setting the Scene
2. **Process Models**
3. Modern Development
4. Maturity Models
5. Good Practices
6. Conclusion

SecAppDev 2019
12

Textbook Example: Microsoft SDL (SD3+C)



Training



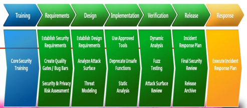
1. **Training**
2. Requirements
3. Design
4. Implementation
5. Verification
6. Release
7. Response

Content

- Secure design
- Threat modeling
- Secure coding
- Security testing
- Privacy

Why?

Requirements


Training | Requirements | Design | Implementation | Verification | Release | Response

Core Security Training

Establish Security Requirements
Create Quality Gates / Bug Bars
Security & Privacy Risk Assessment

Establish Design Requirements
Analyze Attack Surface
Threat Modeling

Use Approved Tools
Deprecate Unsafe Functions
Static Analysis

Dynamic Analysis
Fuzz Testing
Attack Surface Review

Incident Response Plan
Final Security Review
Release Archive

Execute Incident Response Plan

1. Training
2. **Requirements**
3. Design
4. Implementation
5. Verification
6. Release
7. Response

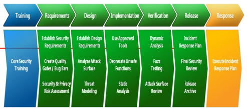
Project inception

When you consider security and privacy at a foundational level

Cost analysis

Determine if development and support costs for improving security and privacy are consistent with business needs

Design


Training | Requirements | Design | Implementation | Verification | Release | Response

Core Security Training

Establish Security Requirements
Create Quality Gates / Bug Bars
Security & Privacy Risk Assessment

Establish Design Requirements
Analyze Attack Surface
Threat Modeling

Use Approved Tools
Deprecate Unsafe Functions
Static Analysis

Dynamic Analysis
Fuzz Testing
Attack Surface Review

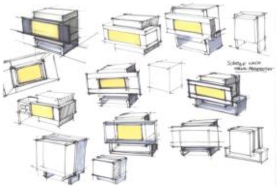
Incident Response Plan
Final Security Review
Release Archive

Execute Incident Response Plan

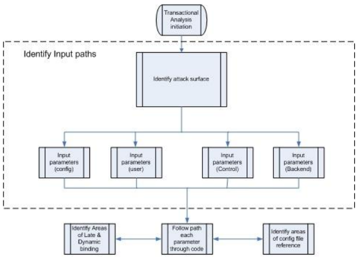
1. Training
2. Requirements
3. **Design**
4. Implementation
5. Verification
6. Release
7. Response

Establish and follow best practices for Design

≠ secure-coding best practices



Risk analysis


Transactional Analysis initiation

Identify Input paths

Identify attack surface

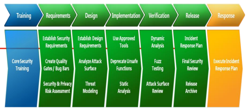
Input parameters (config)
Input parameters (user)
Input parameters (Control)
Input parameters (Backend)

Identify Areas of Late & Dynamic binding
Follow path each parameter through code
Identify areas of config file reference

Threat modeling

STRIDE

Implementation



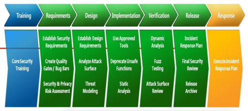
Creating documentation and tools for users that address security and privacy

1. Training
2. Requirements
3. Design
4. **Implementation**
5. Verification
6. Release
7. Response

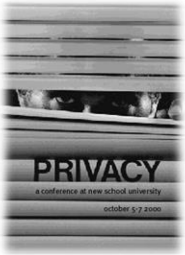
Establish and follow best practices for development

1. Review available information resources
2. Review recommended development tools
3. Define, communicate and document all best practices and policies

Verification



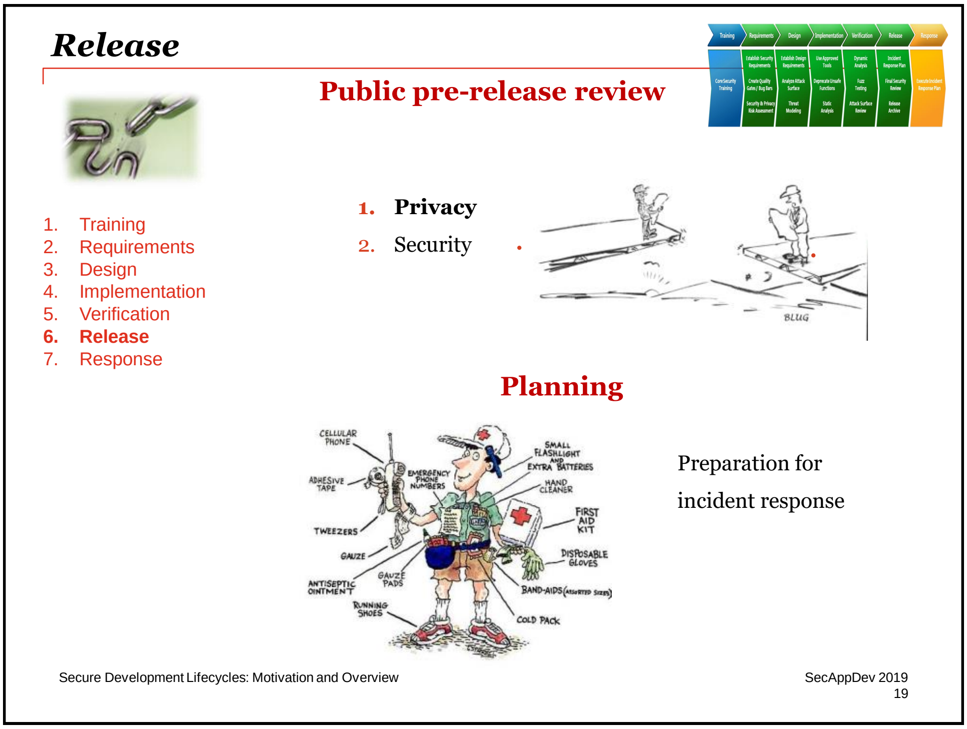
Security and privacy testing



1. Training
2. Requirements
3. Design
4. Implementation
5. **Verification**
6. Release
7. Response

1. Confidentiality, integrity and availability of the software and data processed by the software
2. Freedom from issues that could result in security vulnerabilities

Security push



Release

Training
Requirements
Design
Implementation
Verification
Release
Response
Core Security Training
Establish Security Requirements
Create Quality Gates / Bug Bars
Security & Privacy Risk Assessment
Establish Design Requirements
Analyze Attack Surface
Threat Modeling
Use Approved Tools
Deprecate Unsafe Functions
Static Analysis
Dynamic Analysis
Fuzz Testing
Attack Surface Review
Incident Response Plan
Final Security Review
Release Archive
Execute Incident Response Plan

1. Training
2. Requirements
3. Design
4. Implementation
5. Verification
6. **Release**
7. Response

Public pre-release review

1. **Privacy**
2. Security

BLUG

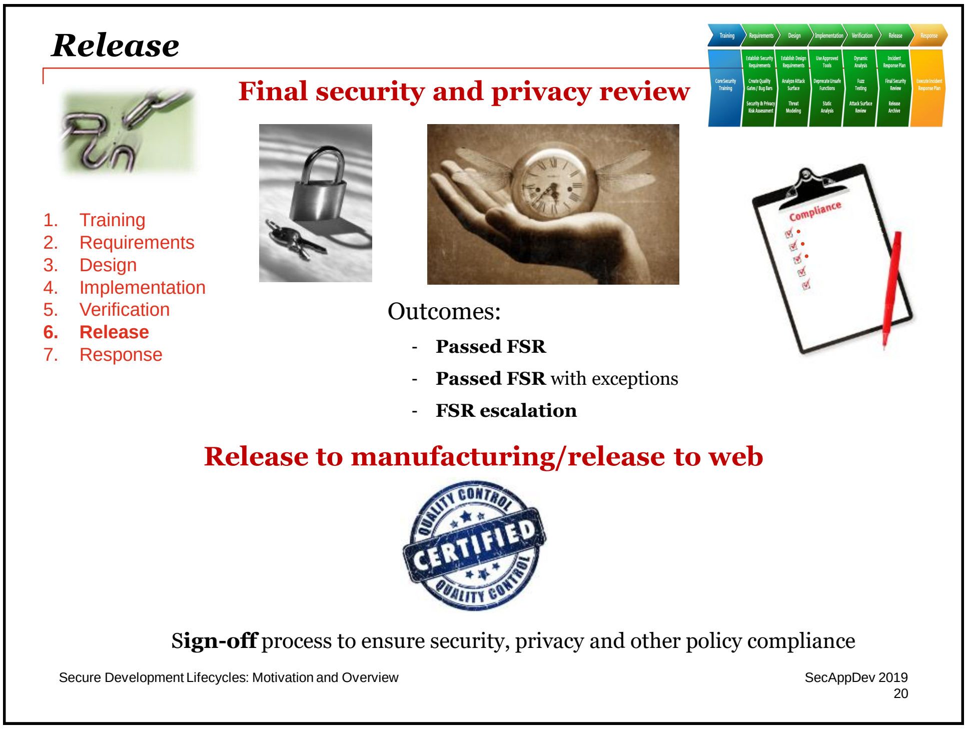
Planning

CELLULAR PHONE
ADHESIVE TAPE
TWEEZERS
GAUZE
ANTISEPTIC OINTMENT
GAUZE PADS
RUNNING SHOES
EMERGENCY PHONE NUMBERS
SMALL FLASHLIGHT AND EXTRA BATTERIES
HAND CLEANER
FIRST AID KIT
DISPOSABLE GLOVES
BAND-AIDS (ASSORTED SIZES)
COLD PACK

Preparation for incident response

Secure Development Lifecycles: Motivation and Overview

SecAppDev 2019
19



Release

Training
Requirements
Design
Implementation
Verification
Release
Response
Core Security Training
Establish Security Requirements
Create Quality Gates / Bug Bars
Security & Privacy Risk Assessment
Establish Design Requirements
Analyze Attack Surface
Threat Modeling
Use Approved Tools
Deprecate Unsafe Functions
Static Analysis
Dynamic Analysis
Fuzz Testing
Attack Surface Review
Incident Response Plan
Final Security Review
Release Archive
Execute Incident Response Plan

1. Training
2. Requirements
3. Design
4. Implementation
5. Verification
6. **Release**
7. Response

Final security and privacy review

Compliance

Outcomes:

- **Passed FSR**
- **Passed FSR** with exceptions
- **FSR escalation**

Release to manufacturing/release to web

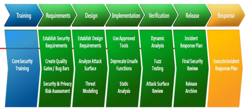
QUALITY CONTROL
CERTIFIED
QUALITY CONTROL

Sign-off process to ensure security, privacy and other policy compliance

Secure Development Lifecycles: Motivation and Overview

SecAppDev 2019
20

Response



Training | Requirements | Design | Implementation | Verification | Release | Response

Core Security Training

Establish Security Requirements
Create Quality Gates / Bug Bars
Security & Privacy Risk Assessment

Establish Design Requirements
Analyze Attack Surface
Threat Modeling

Use Approved Tools
Deprecate Unsafe Functions
Static Analysis

Dynamic Analysis
Fuzz Testing
Attack Surface Review

Incident Response Plan
Final Security Review
Release Archive

Execute Incident Response Plan

Execute Incident Response Plan

1. Training
2. Requirements
3. Design
4. Implementation
5. Verification
6. Release
7. **Response**

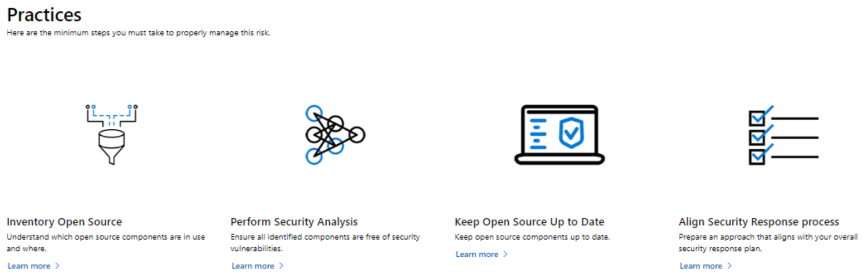
=> able to respond appropriately to reports of vulnerabilities in their software products, and to attempted exploitation of those vulnerabilities.

Microsoft SDL Practices (Anno 2019)

1. Provide Training
2. Define Security Requirements
3. Define Metrics and Compliance Reporting
4. Perform Threat Modeling
5. Establish Design Requirements
6. Define and Use Cryptographic Standards
7. Manage the Security Risk of Using Third-Party Components
8. Use Approved Tools
9. Perform Static Analysis Security Testing (SAST)
10. Perform Dynamic Analysis Security Testing (DAST)
11. Perform Penetration Testing
12. Establish a Standard Incident Response Process

Source: www.microsoft.com

Microsoft SDL Example - Using Open Source



Practices

Here are the minimum steps you must take to properly manage this risk.

Inventory Open Source
Understand which open source components are in use and where.
Learn more >

Perform Security Analysis
Ensure all identified components are free of security vulnerabilities.
Learn more >

Keep Open Source Up to Date
Keep open source components up to date.
Learn more >

Align Security Response process
Prepare an approach that aligns with your overall security response plan.
Learn more >

Source: www.microsoft.com

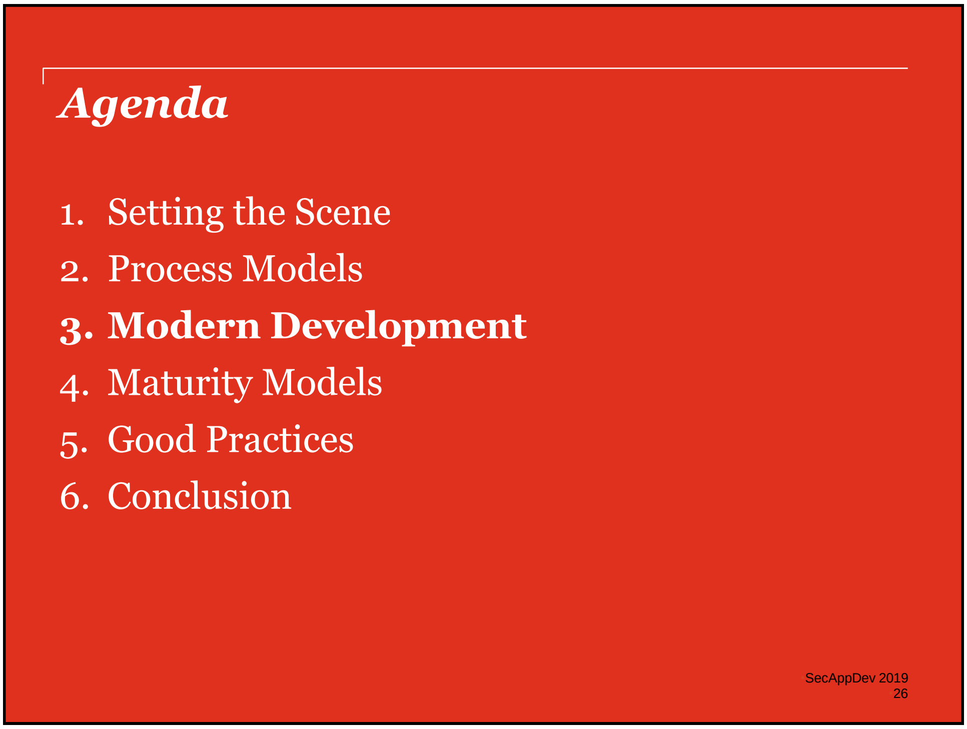
Final note on process models

Process models provide a good starting point into secure development lifecycles

- Overview of different activities that are relevant
- Indication of ordering and dependencies

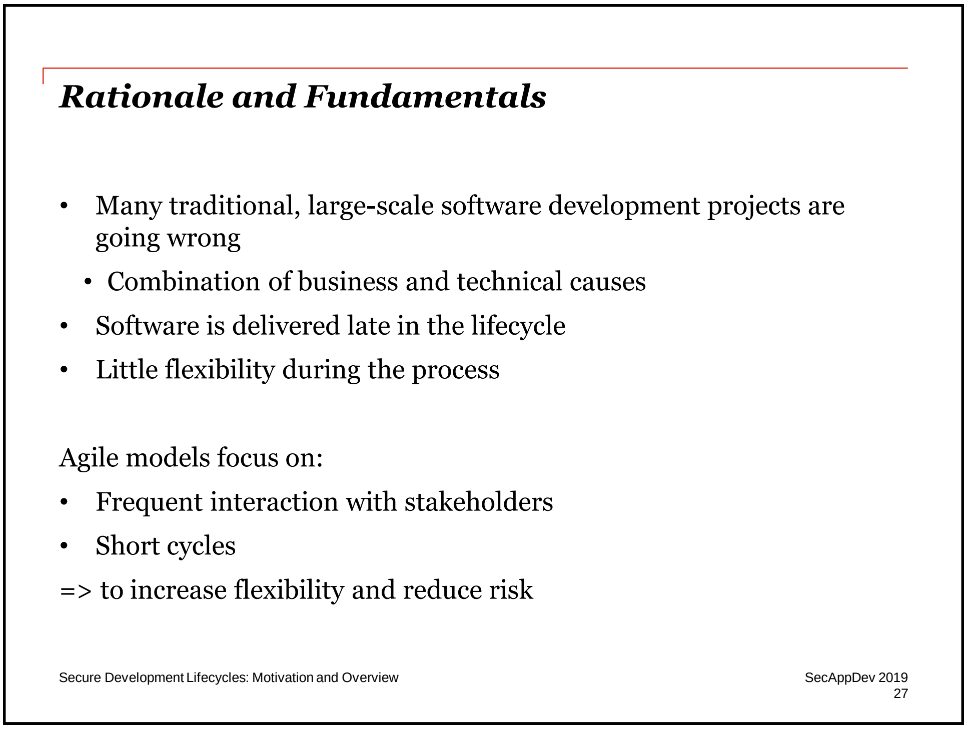
Only few companies still work using a traditional, waterfall-only paradigm

- Process models will not suffice for modern development environments
- Need to be complemented (or replaced) with other techniques to be useful



Agenda

1. Setting the Scene
2. Process Models
3. **Modern Development**
4. Maturity Models
5. Good Practices
6. Conclusion



Rationale and Fundamentals

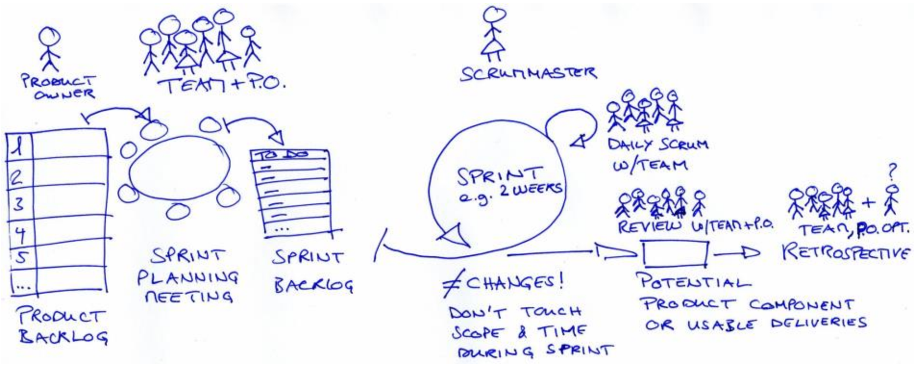
- Many traditional, large-scale software development projects are going wrong
  - Combination of business and technical causes
- Software is delivered late in the lifecycle
- Little flexibility during the process

Agile models focus on:

- Frequent interaction with stakeholders
- Short cycles

=> to increase flexibility and reduce risk

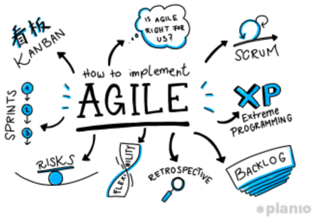
Scrum

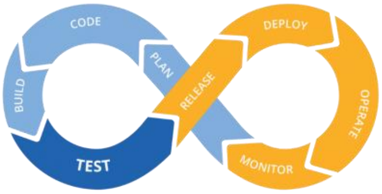


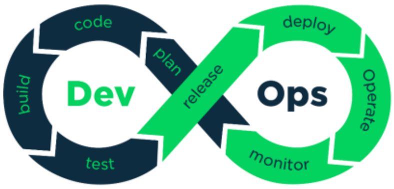
Modern & Secure development: a mismatch ?

Agile Dev.	Security
Speed & Flexibility	Stable & Rigorous
Short cycles	Extra activities
Limited documentation	Extensive analysis
Functionality-driven	Non-functional

Modern Development





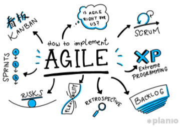


Secure Modern Development
General principles

- Make security a natural part of the lifecycle, but don’t overdo
  - lightweight, in-phase and iterative
  - Preventive, detective and reactive controls

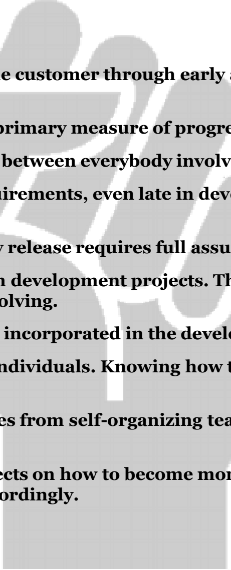
- Be involved at key moments in the lifecycle
- Automate security
- Work on established concepts & practices
- Continuous testing
- Small steps at a time (i.e. continuous improvement)

Security Tactics for Agile Development

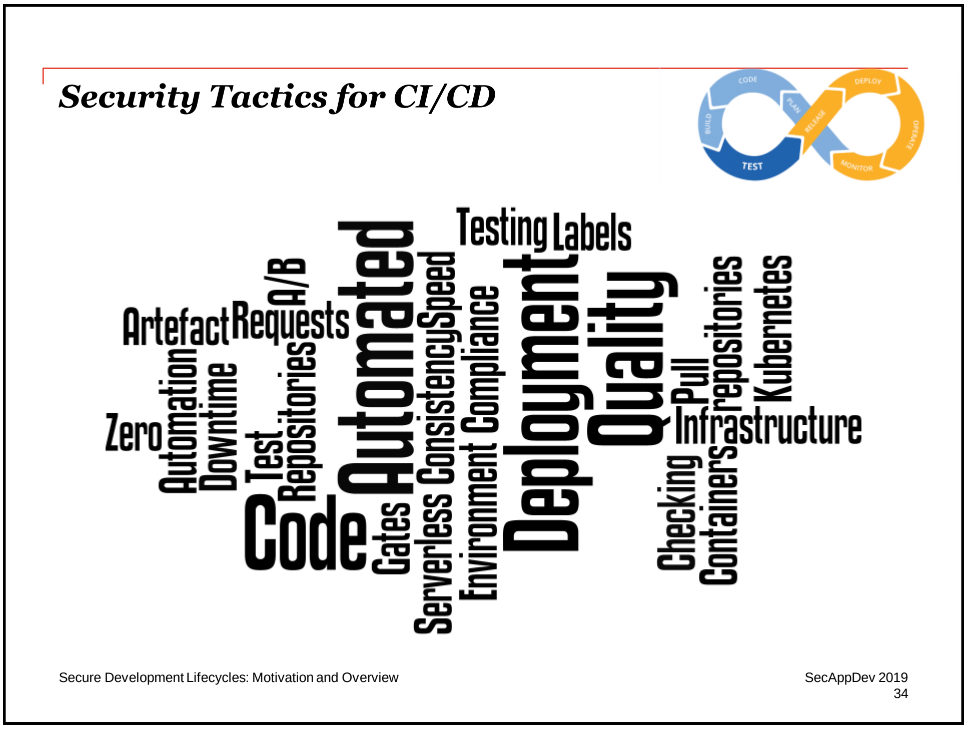


Deploy
User
Sprint
Modeling
Model
Retrospective
Zero
Security
Testing
Done
Incremental
Dare
Planning
Definition
Stories
Iteration
Acceptance
Champion
Abuser
Criteria
Spotify
Threat

Secure Agile Manifesto



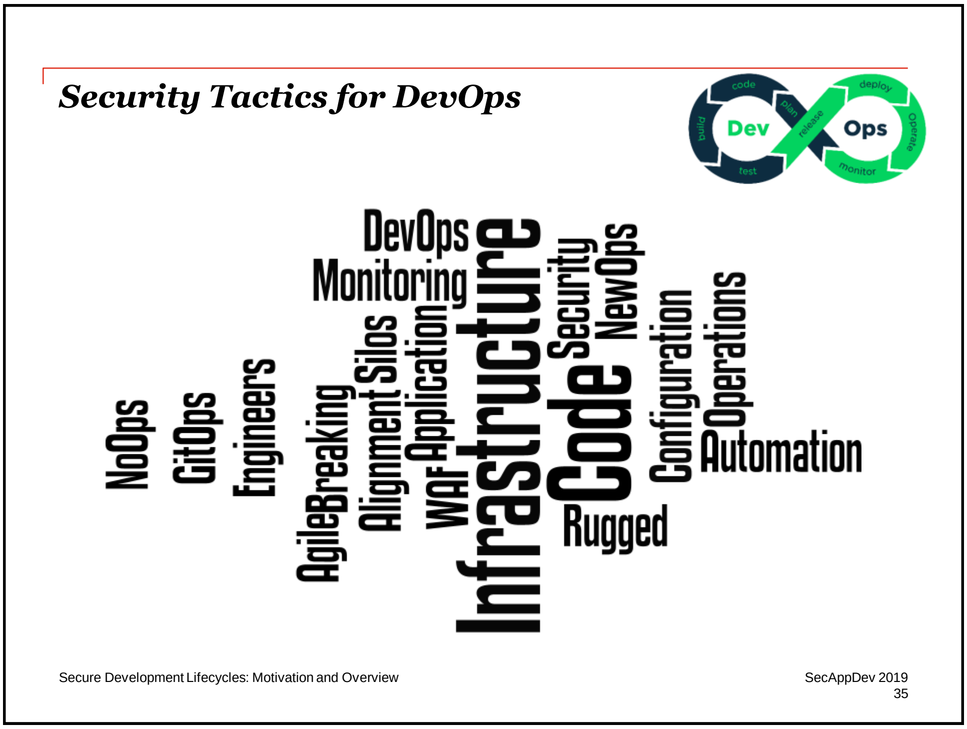
1. **Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.**
2. **Working valuable software is the primary measure of progress.**
3. **Security is a shared responsibility between everybody involved in the life cycle of the product.**
4. **Welcome changing (security) requirements, even late in development, taking into account that enough security is enough.**
5. **Dare to deploy software. Not every release requires full assurance.**
6. **Provide security elements to use in development projects. These elements should be known, readily available and continuously evolving.**
7. **Security should be automated and incorporated in the development practices.**
8. **Build projects around motivated individuals. Knowing how to build secure software is an intrinsic motivator.**
9. **The most effective solution emerges from self-organizing teams able to call upon security experts when needed.**
10. **At regular intervals, the team reflects on how to become more effective, adjusting its processes and technical solutions accordingly.**



Security Tactics for CI/CD

Secure Development Lifecycles: Motivation and Overview

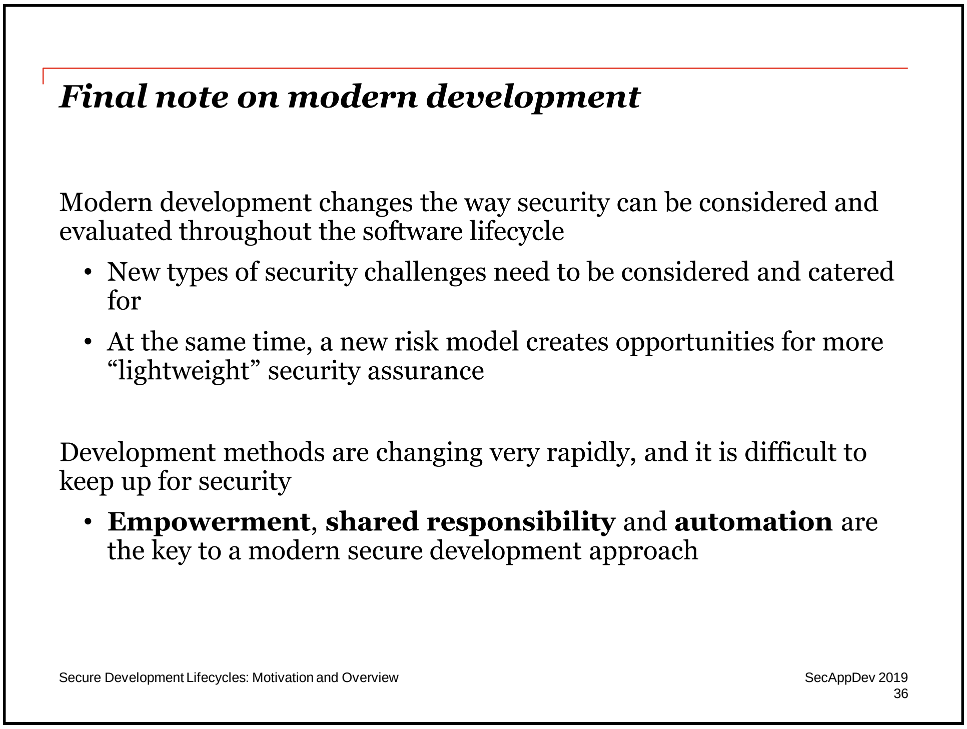
SecAppDev 2019
34



Security Tactics for DevOps

Secure Development Lifecycles: Motivation and Overview

SecAppDev 2019
35



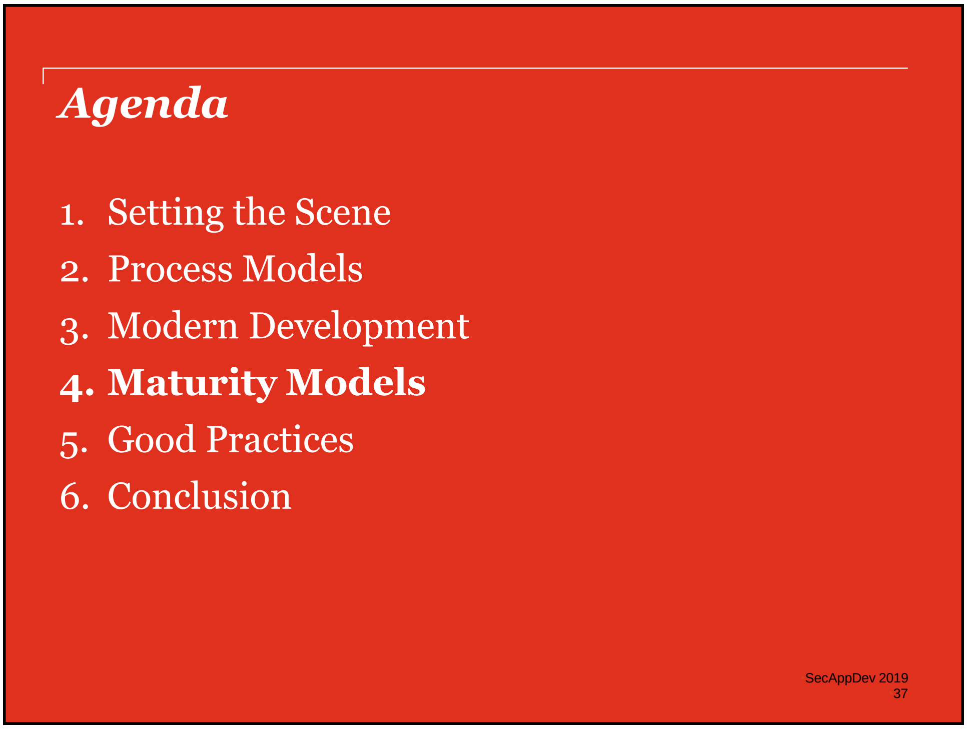
Final note on modern development

Modern development changes the way security can be considered and evaluated throughout the software lifecycle

- New types of security challenges need to be considered and catered for
- At the same time, a new risk model creates opportunities for more “lightweight” security assurance

Development methods are changing very rapidly, and it is difficult to keep up for security

- **Empowerment**, **shared responsibility** and **automation** are the key to a modern secure development approach



Agenda

1. Setting the Scene
2. Process Models
3. Modern Development
4. **Maturity Models**
5. Good Practices
6. Conclusion

Why Maturity Models for SDLC?

An organization’s behavior changes slowly over time.

- Changes must be **iterative** while working toward long-term goals

There is no single recipe that works for all organizations

- A solution must enable **risk-based** choices tailored to the organization

Guidance related to security activities must be prescriptive

- A solution must provide enough details for non-security-people

Overall, must be simple, well-defined, and **measurable**

To answer questions like

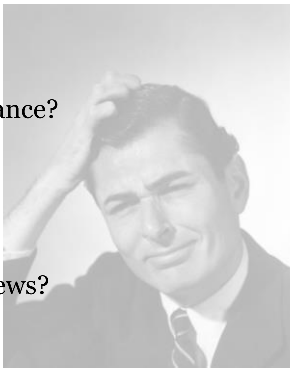
What should we be doing in our SDLC?

What are others doing in terms of software assurance?

What are good practices for software assurance?

Should we focus on threat modelling or code reviews?

How much time/effort/cost will this take?



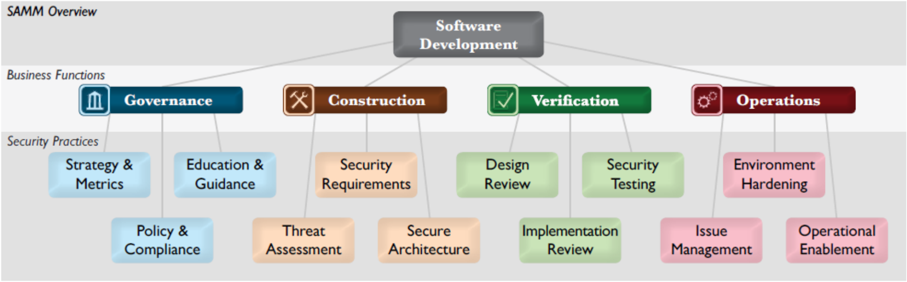
Textbook Example: OWASP SAMM

Scope: Entire software lifecycle, rather than just development.

https://owaspsamm.org/

Version 1.5, 2017

Core Structure



Notion of Maturity

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

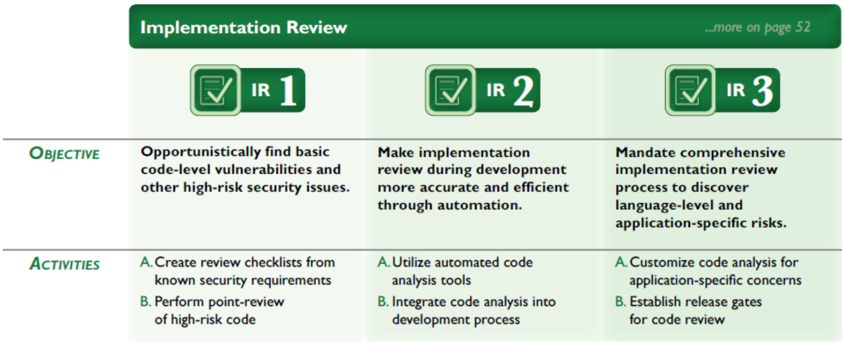
1 Initial understanding and adhoc provision of security practice

2 Increase efficiency and/or effectiveness of the security practice

3 Comprehensive mastery of the security practice at scale

CMMI ?

An example



Implementation Review ...more on page 52

	IR 1	IR 2	IR 3
OBJECTIVE	Opportunistically find basic code-level vulnerabilities and other high-risk security issues.	Make implementation review during development more accurate and efficient through automation.	Mandate comprehensive implementation review process to discover language-level and application-specific risks.
ACTIVITIES	A. Create review checklists from known security requirements B. Perform point-review of high-risk code	A. Utilize automated code analysis tools B. Integrate code analysis into development process	A. Customize code analysis for application-specific concerns B. Establish release gates for code review

SAMM also defines

Objective

Activities

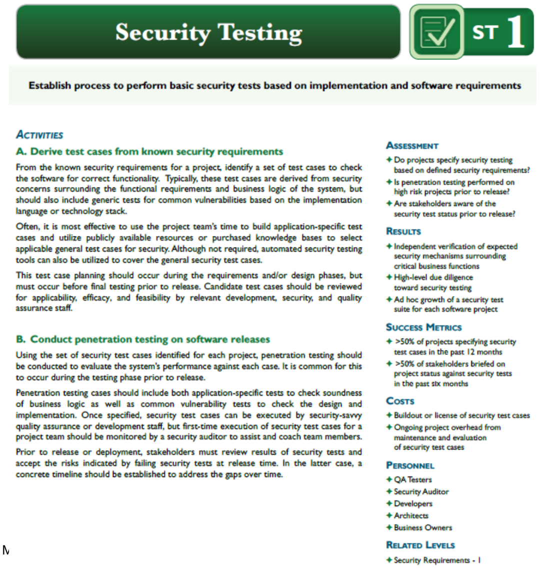
Results

Success Metrics

Costs

Personnel

Related Levels



Security Testing

ST 1

Establish process to perform basic security tests based on implementation and software requirements

Activities

A. Derive test cases from known security requirements

From the known security requirements for a project, identify a set of test cases to check the software for correct functionality. Typically, these test cases are derived from security concerns surrounding the functional requirements and business logic of the system, but should also include generic tests for common vulnerabilities based on the implementation language or technology stack.

Often, it is most effective to use the project team's time to build application-specific test cases and utilize publicly available resources or purchased knowledge bases to select applicable general test cases for security. Although not required, automated security testing tools can also be utilized to cover the general security test cases.

This test case planning should occur during the requirements and/or design phases, but must occur before final testing prior to release. Candidate test cases should be reviewed for applicability, efficacy, and feasibility by relevant development, security, and quality assurance staff.

B. Conduct penetration testing on software releases

Using the set of security test cases identified for each project, penetration testing should be conducted to evaluate the system's performance against each case. It is common for this to occur during the testing phase prior to release.

Penetration testing cases should include both application-specific tests to check soundness of business logic as well as common vulnerability tests to check the design and implementation. Once specified, security test cases can be executed by security-savvy quality assurance or development staff, but first-time execution of security test cases for a project team should be monitored by a security auditor to assist and coach team members.

Prior to release or deployment, stakeholders must review results of security tests and accept the risks indicated by failing security tests at release time. In the latter case, a concrete timeline should be established to address the gaps over time.

Assessment

- Do projects specify security testing based on defined security requirements?
- Is penetration testing performed on high risk projects prior to release?
- Are stakeholders aware of the security test status prior to release?

Results

- Independent verification of expected security mechanisms surrounding critical business functions
- High-level due diligence toward security testing
- Ad hoc growth of a security test suite for each software project

Success Metrics

- >50% of projects specifying security test cases in the past 12 months
- >50% of stakeholders briefed on project status against security tests in the past six months

Costs

- Buildout or license of security test cases
- Ongoing project overhead from maintenance and evaluation of security test cases

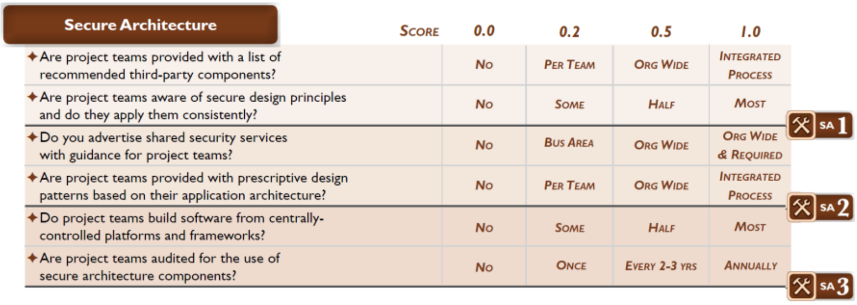
Personnel

- QA Testers
- Security Auditor
- Developers
- Architects
- Business Owners

Related Levels

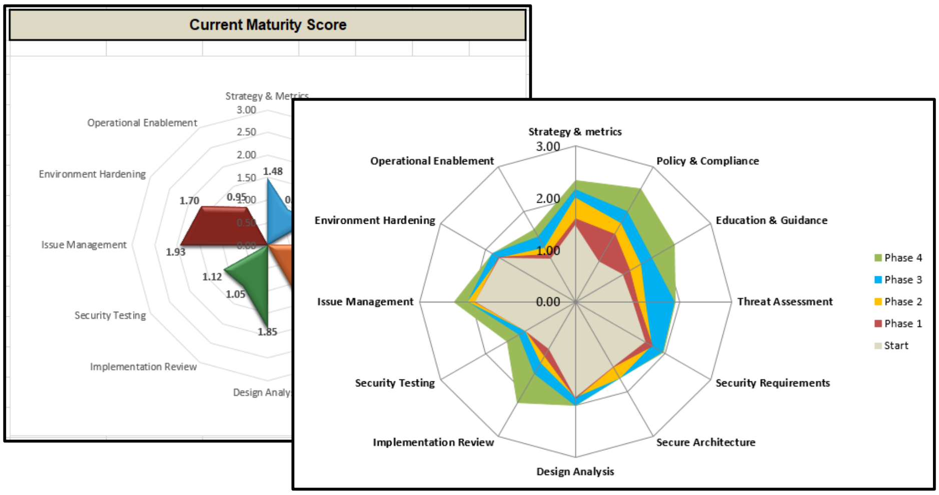
- Security Requirements - 1

Conducting assessments



Secure Architecture	Score	0.0	0.2	0.5	1.0	
✦ Are project teams provided with a list of recommended third-party components?		No	Per Team	Org Wide	Integrated Process	
✦ Are project teams aware of secure design principles and do they apply them consistently?		No	Some	Half	Most	SA 1
✦ Do you advertise shared security services with guidance for project teams?		No	Bus Area	Org Wide	Org Wide & Required	
✦ Are project teams provided with prescriptive design patterns based on their application architecture?		No	Per Team	Org Wide	Integrated Process	SA 2
✦ Do project teams build software from centrally-controlled platforms and frameworks?		No	Some	Half	Most	
✦ Are project teams audited for the use of secure architecture components?		No	Once	Every 2-3 yrs	Annually	SA 3

Assessments and Roadmaps

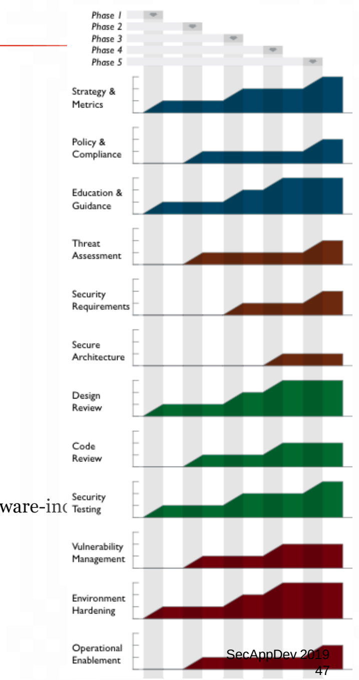


Secure Development Lifecycles: Motivation and Overview

SecAppDev 2019
46

Roadmap templates

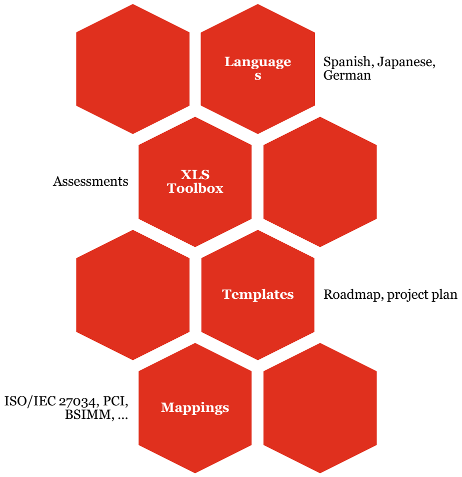
- To make the “building blocks” usable, SAMM defines Roadmaps templates for typical kinds of organizations
  - Independent Software Vendors
  - Online Service Providers
  - Financial Services Organizations
  - Government Organizations
- Organization types chosen because
  - They represent common use-cases
  - Each organization has variations in typical software-inc
  - Optimal creation of an assurance program is different for each



•Secure Development Lifecycles: Motivation and Overview

SecAppDev 2019
47

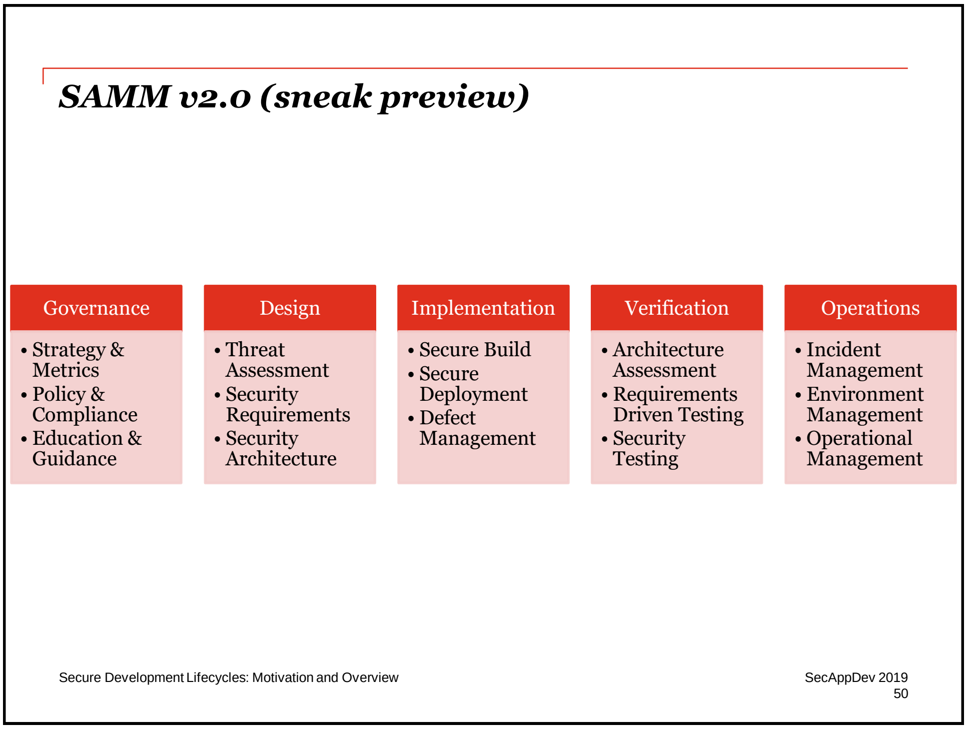
SAMM Tools



Challenges in v1.5

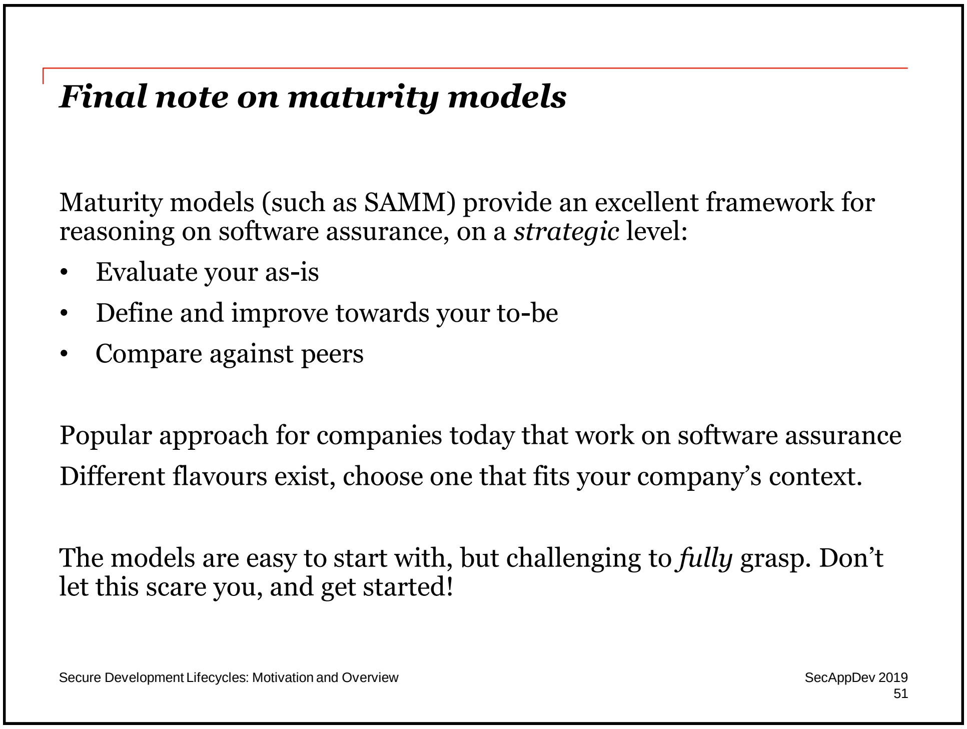
- Waterfall-like setup
- Lacking activities/perspectives
- Logical flow between activities
- Measuring quality of implementation (vs. coverage)
- Lack of a good dataset

These will be tackled in the **upcoming v2.0** of the model (foreseen for Summer 2019).



SAMM v2.0 (sneak preview)

Governance	Design	Implementation	Verification	Operations
• Strategy & Metrics • Policy & Compliance • Education & Guidance	• Threat Assessment • Security Requirements • Security Architecture	• Secure Build • Secure Deployment • Defect Management	• Architecture Assessment • Requirements Driven Testing • Security Testing	• Incident Management • Environment Management • Operational Management



Final note on maturity models

Maturity models (such as SAMM) provide an excellent framework for reasoning on software assurance, on a *strategic* level:

- Evaluate your as-is
- Define and improve towards your to-be
- Compare against peers

Popular approach for companies today that work on software assurance

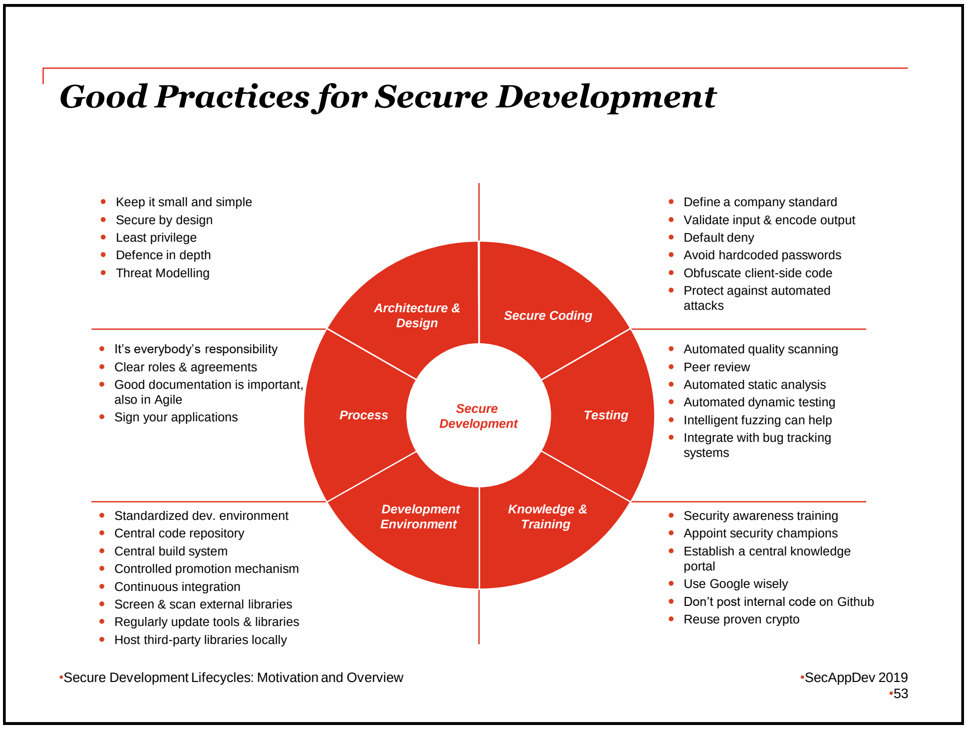
Different flavours exist, choose one that fits your company’s context.

The models are easy to start with, but challenging to *fully* grasp. Don’t let this scare you, and get started!

Agenda

1. Setting the Scene
2. Process Models
3. Modern Development
4. Maturity Models
5. **Good Practices**
6. Conclusion

SecAppDev 2019
52



Good Practices for Secure Development

Architecture & Design

- Keep it small and simple
- Secure by design
- Least privilege
- Defence in depth
- Threat Modelling

Secure Coding

- Define a company standard
- Validate input & encode output
- Default deny
- Avoid hardcoded passwords
- Obfuscate client-side code
- Protect against automated attacks

Process

- It's everybody's responsibility
- Clear roles & agreements
- Good documentation is important, also in Agile
- Sign your applications

Secure Development

Testing

- Automated quality scanning
- Peer review
- Automated static analysis
- Automated dynamic testing
- Intelligent fuzzing can help
- Integrate with bug tracking systems

Development Environment

- Standardized dev. environment
- Central code repository
- Central build system
- Controlled promotion mechanism
- Continuous integration
- Screen & scan external libraries
- Regularly update tools & libraries
- Host third-party libraries locally

Knowledge & Training

- Security awareness training
- Appoint security champions
- Establish a central knowledge portal
- Use Google wisely
- Don't post internal code on Github
- Reuse proven crypto

•Secure Development Lifecycles: Motivation and Overview

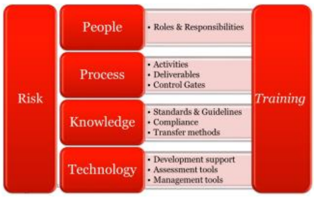
•SecAppDev 2019
•53

Agenda

1. Setting the Scene
2. Process Models
3. Modern Development
4. Maturity Models
5. Good Practices
6. **Conclusion**

Summary and key take-aways

- Secure development is …
  - in the eye of the beholder
  - everybody’s responsibility



Risk

People
- Roles & Responsibilities

Process
- Activities
- Deliverables
- Control Gates

Knowledge
- Standards & Guidelines
- Compliance
- Transfer methods

Technology
- Development support
- Assessment tools
- Management tools

Training

- SDLC flavours exist for traditional and modern development methods
- Maturity models can help in reasoning about progress

- All models need to be adapted and fine-tuned to your organisation’s context and culture to become effective
  - This is the challenging part
  - It’s a continuous journey

Questions ?

Secure Development Lifecycles: Motivation and Overview

SecAppDev 2019
56