www.pwc.com

Secure Development LifeCycles (SDLC)

Bart De Win

Feb 2014

SecAppDev 2014

pwc

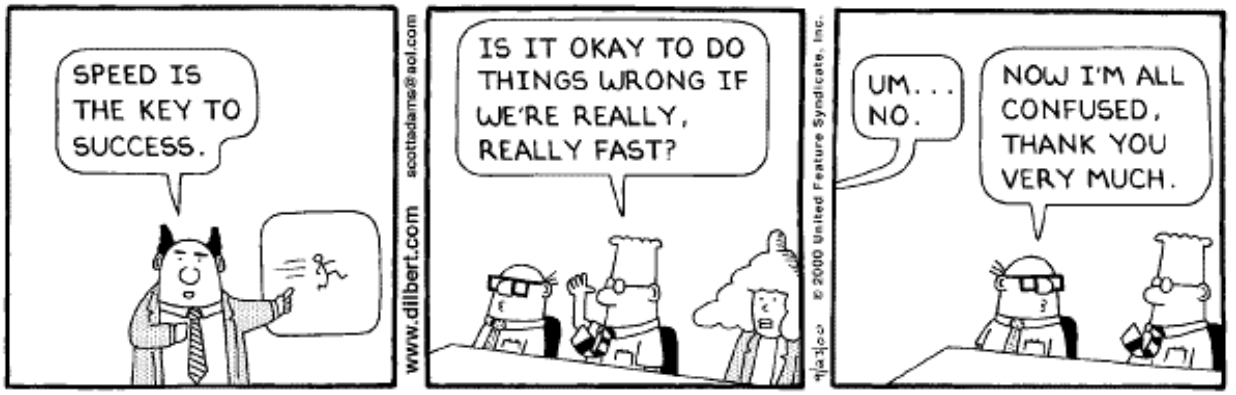
Bart De Win ?

- 15+ years of Information Security Experience
  - Ph.D. in Computer Science - Application Security
- Author of >60 scientific publications
- ISC² CSSLP certified
- Senior Manager @ PwC Belgium:
  - Expertise Center Leader Secure Software
  - (Web) Application tester (pentesting, arch. review, code review, …)
  - Trainer for several courses related to secure software
  - Specialized in Secure Software Development Lifecycle (SDLC)
- OWASP OpenSAMM co-leader
- Contact me at bart.de.win@be.pwc.com

Agenda

1. **Motivation**
2. Process Models
3. Maturity Models
4. Agile Development
5. Conclusion

Application Security Problem



Adaptability

Technology stacks

Software complexity

Training

Better

Growing connectivity

Mobile

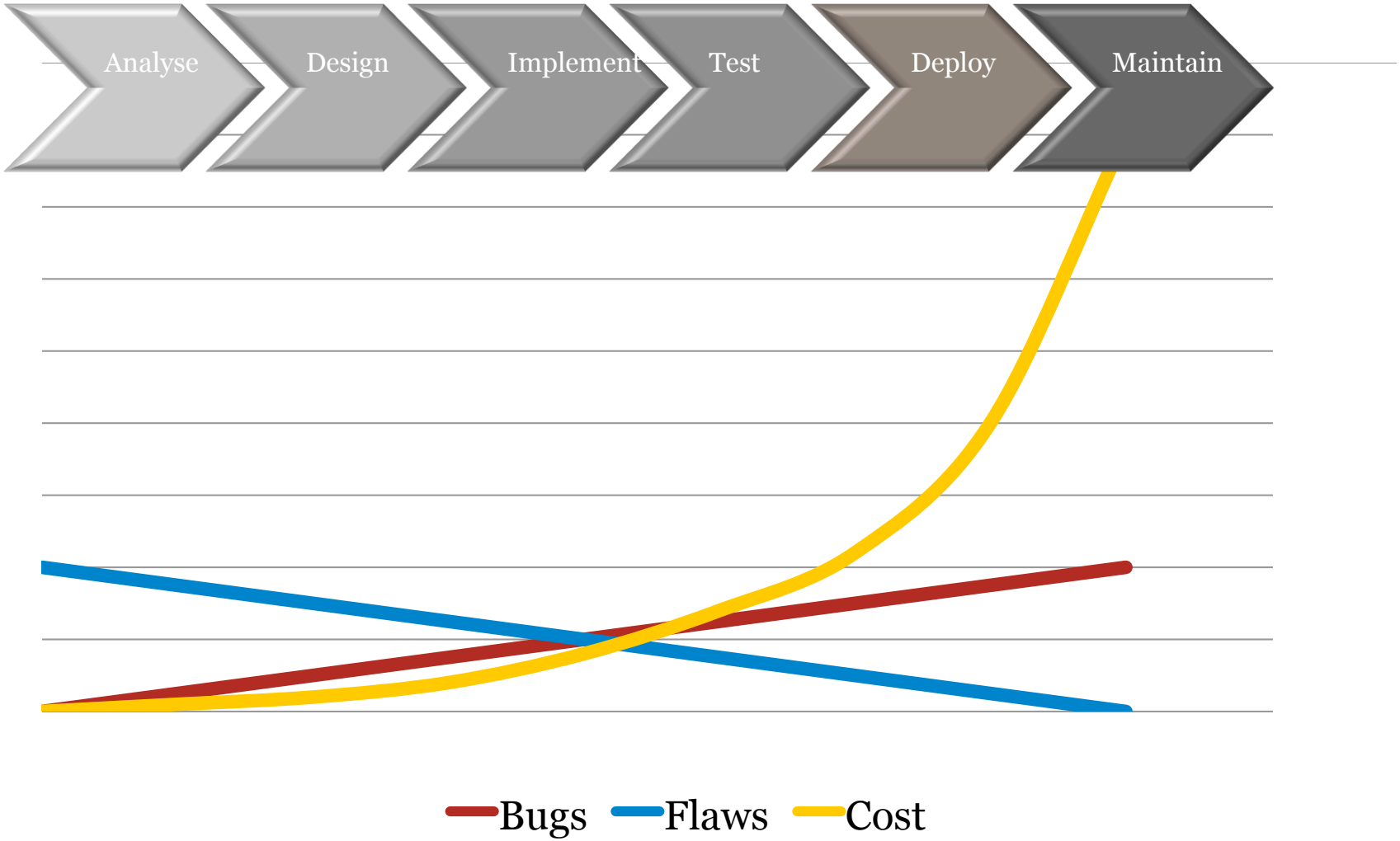
Faster

Cloud

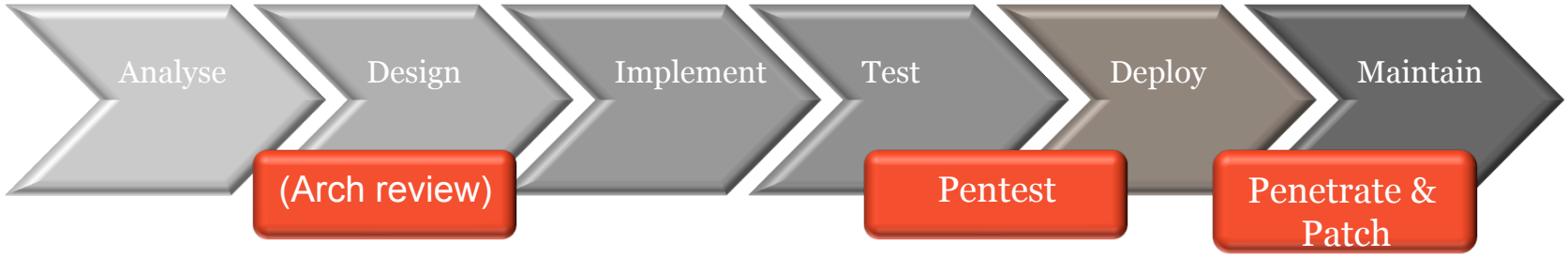
75% of vulnerabilities are application related

Application Security Symbiosis

Application Security during Software Development



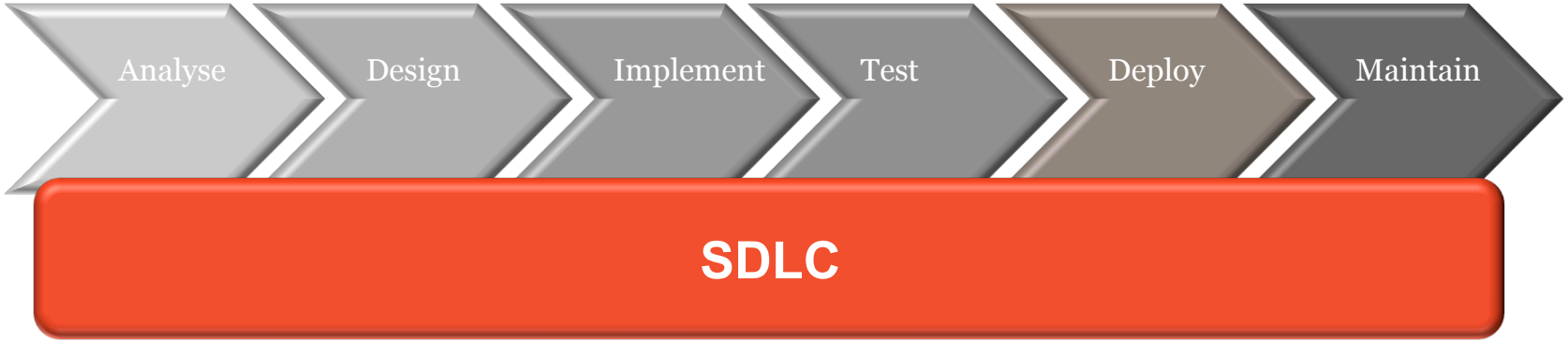
The State-of-Practice in Secure Software Development



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

SDLC ?



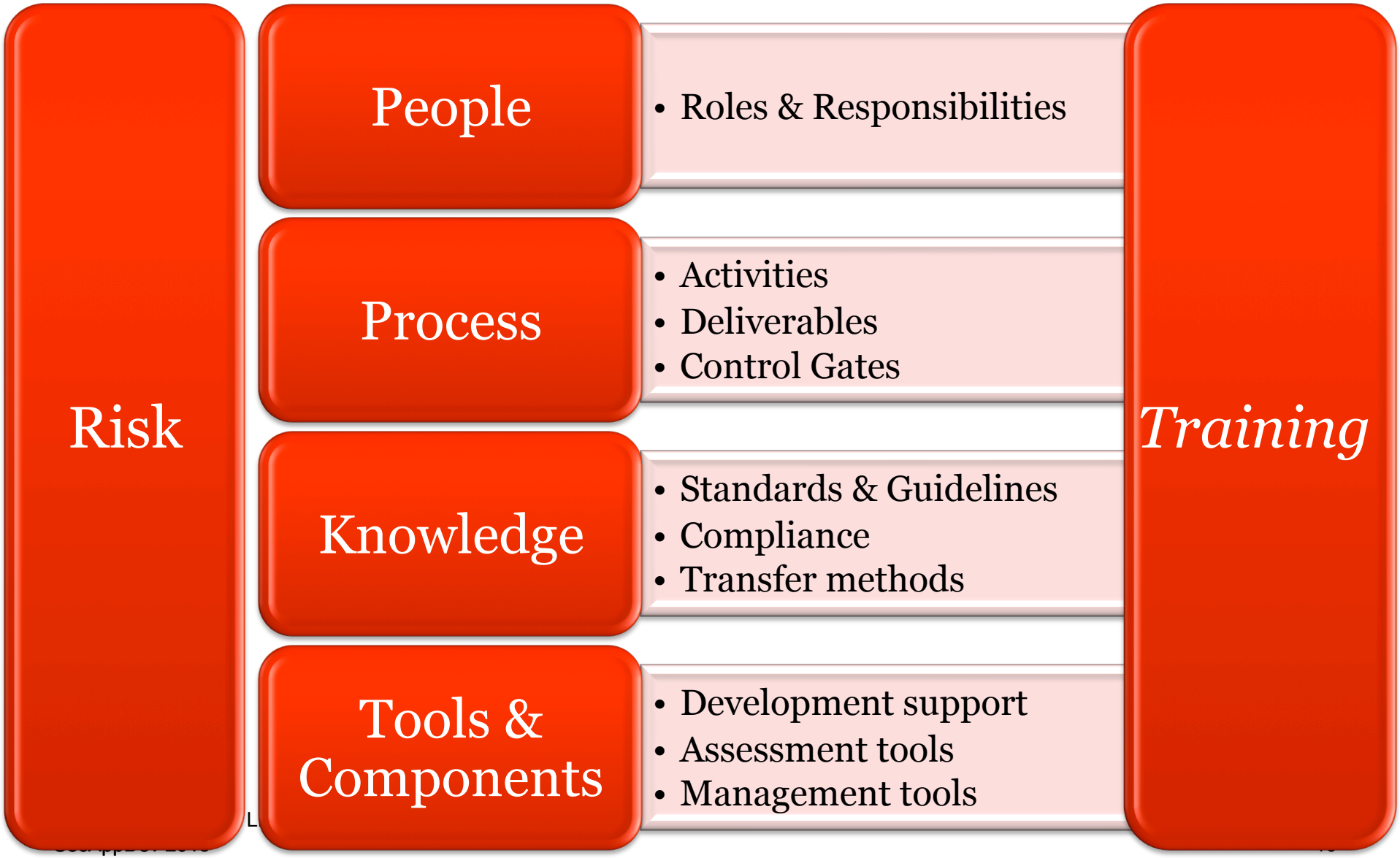
Enterprise-wide software security improvement program

- Strategic approach to assure software quality
- Goal is to increase systematicity
- Focus on security functionality and security hygiene

SDLC Objectives

To develop (and maintain) software in a **consistent** and **efficient** way with a **demonstrable** & **standards-compliant security quality**, inline with the organizational **risks**.

SDLC Cornerstones

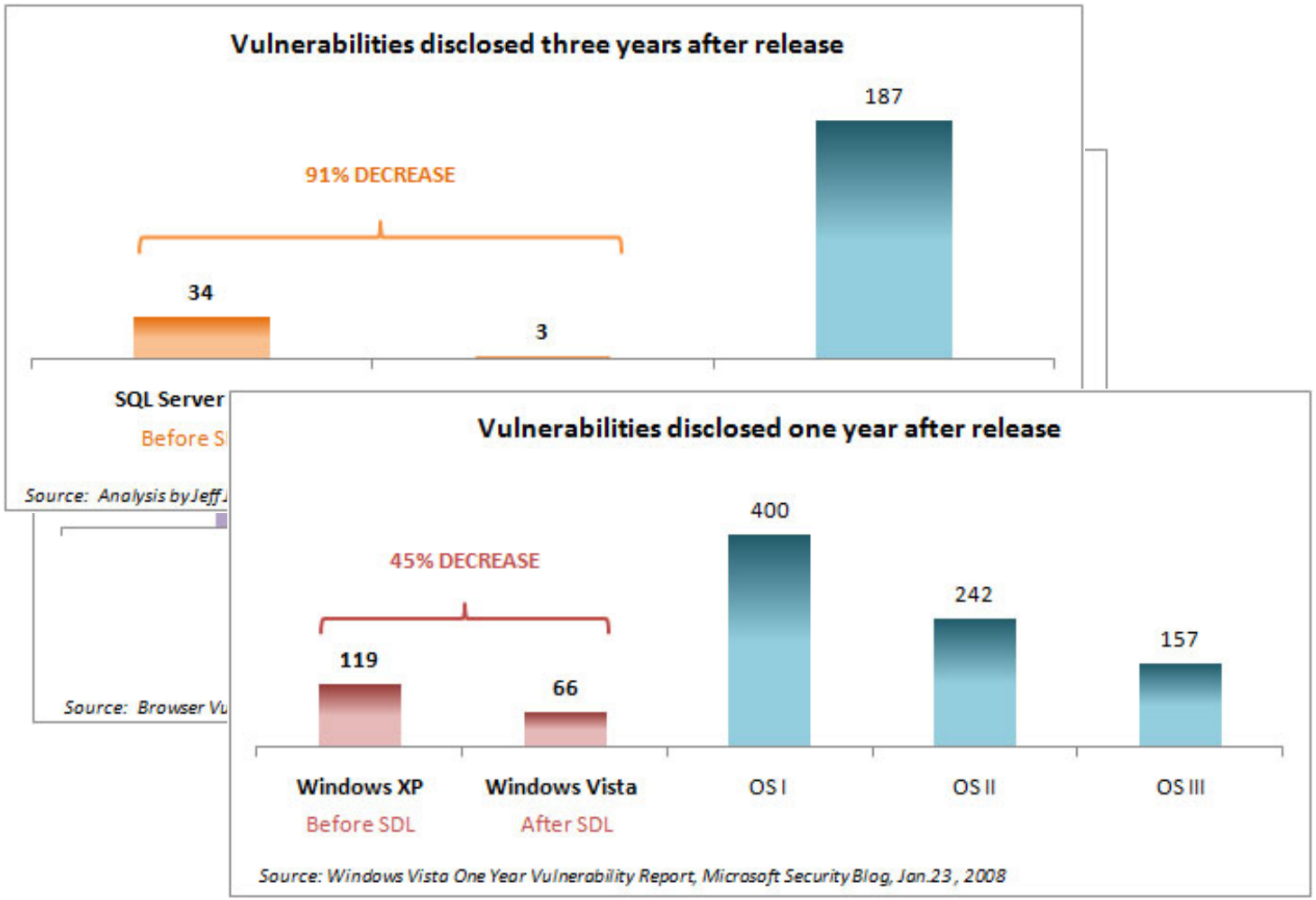


Strategic ?

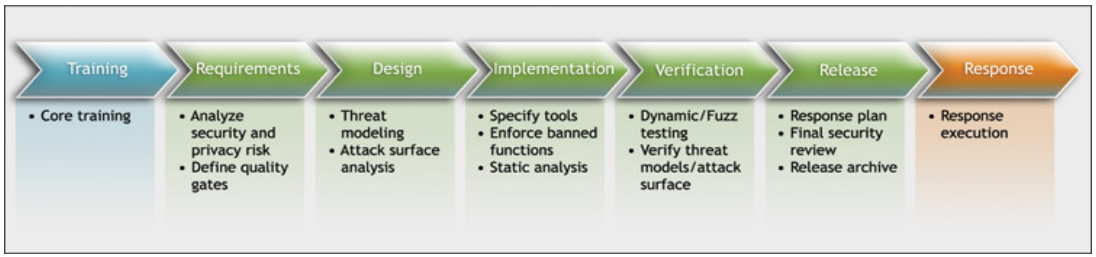
Organizations with a proper SDLC will experience an 80 percent decrease in critical vulnerabilities

Organizations that acquire products and services with just a 50 percent reduction in vulnerabilities will reduce configuration management and incident response costs by 75 percent each.

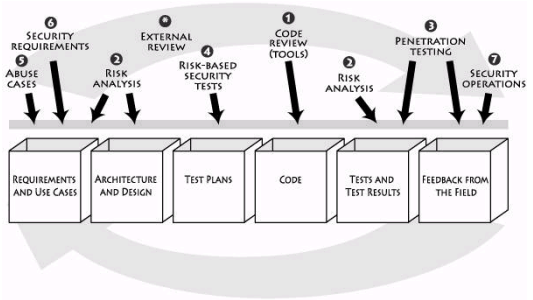
Does it really work ?



(Some) SDLC-related initiatives



•Microsoft SDL

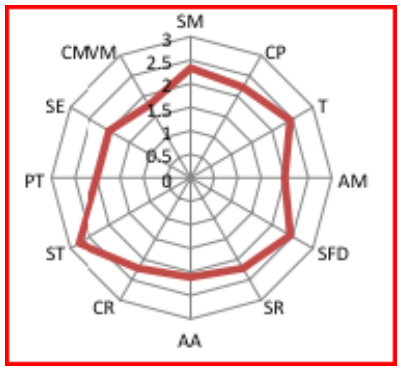


•TouchPoints

•CLASP

NIST National Institute of Standards and Technology

•SP800-64



•BSIMM

SAFECode Software Assurance Forum for Excellence in Code Driving Security and Integrity

SSE CMM

•SSE-CMM

Gartner

Software Engineering Institute | Carnegie Mellon

•TSP-Secure

Software Assurance Maturity Model
A guide to building security into software development
Version - 1.0

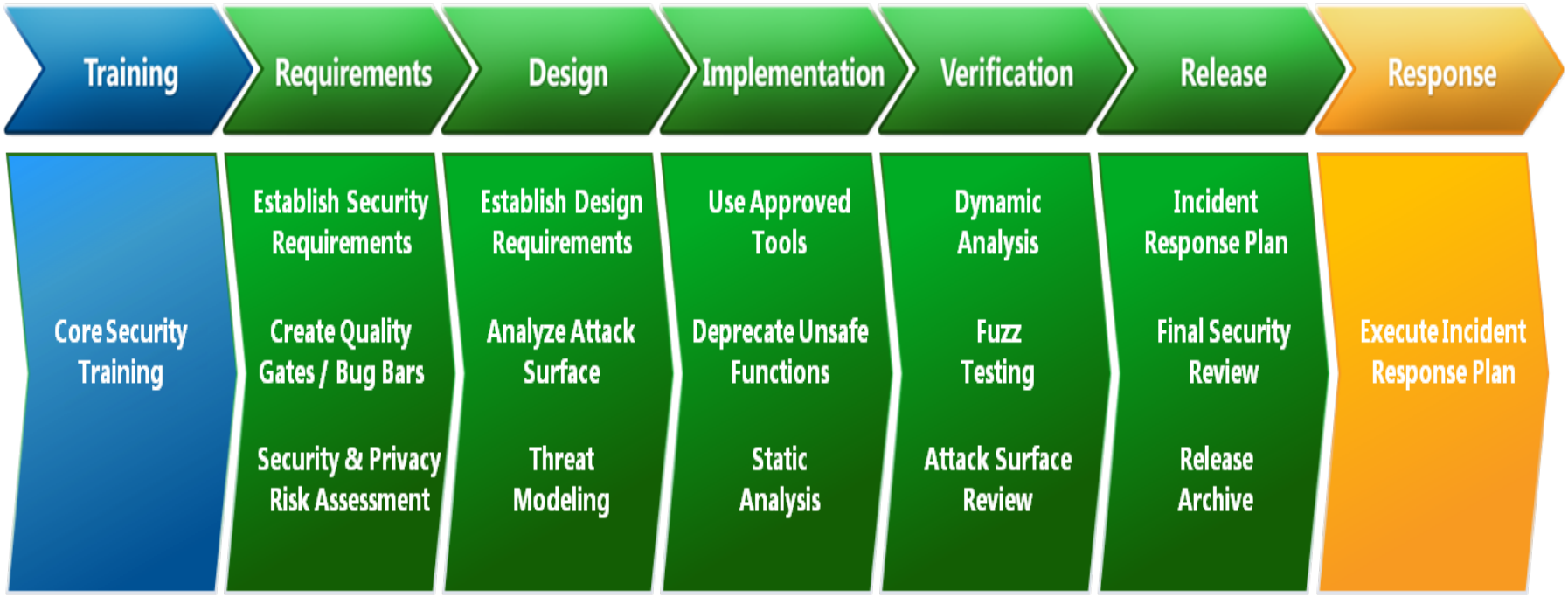
•SAMM

•GASSP

Agenda

1. Motivation
2. **Process Models**
3. Maturity Models
4. Agile Development
5. Conclusion

Selected Example: Microsoft SDL (SD3+C)



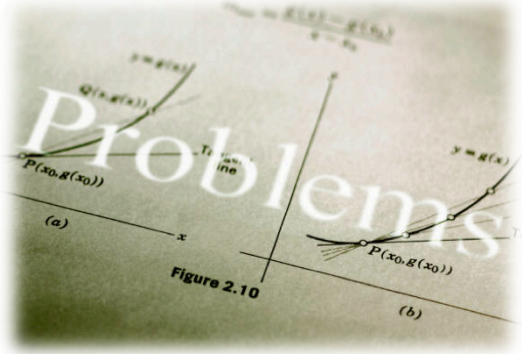
Training

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

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

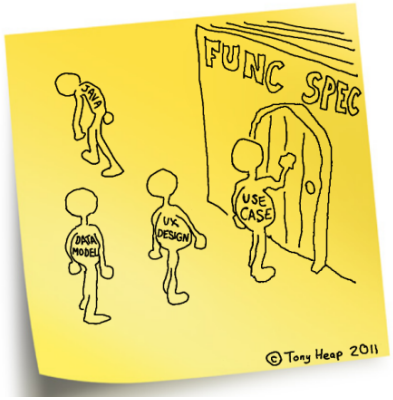
Cost analysis

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

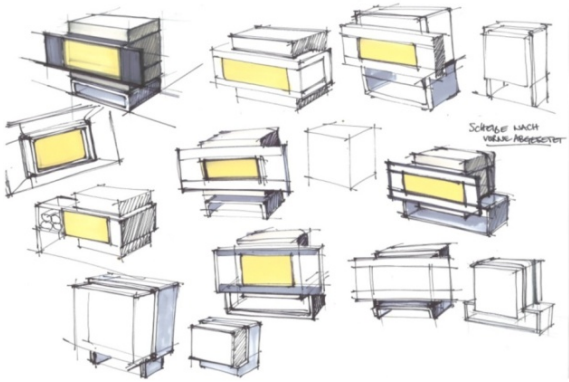
Design

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

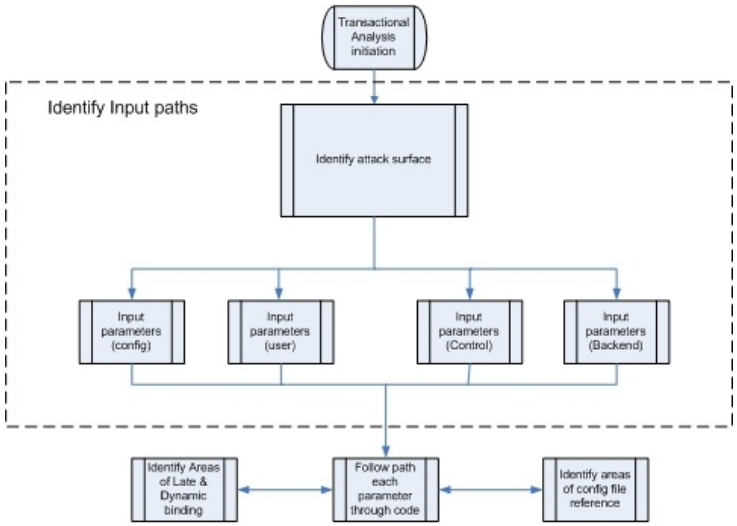
Establish and follow best practices for Design



≠ secure-coding best practices



Risk analysis



Threat modeling

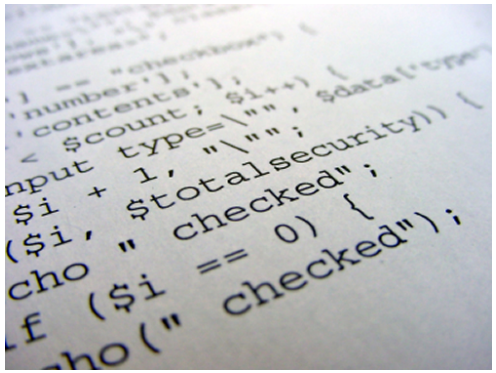
STRIDE

Implementation

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

Creating documentation and tools for users that address security and privacy

Establish and follow best practices for development

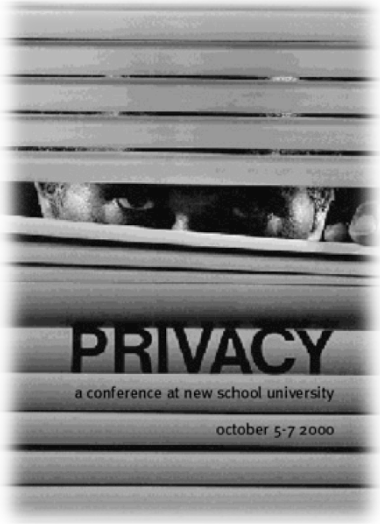


1. Review available information resources
2. Review recommended development tools
3. Define, communicate and document all best practices and policies

Verification

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

Security and privacy testing



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

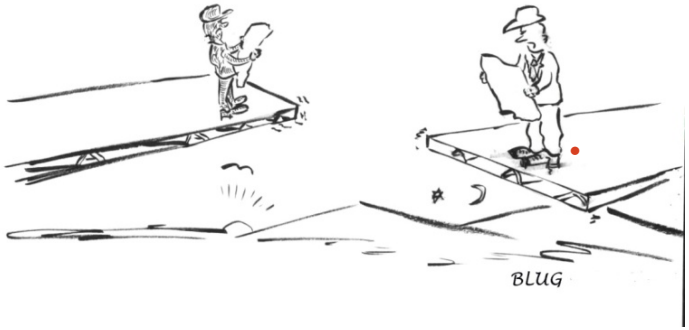
Security push

Release

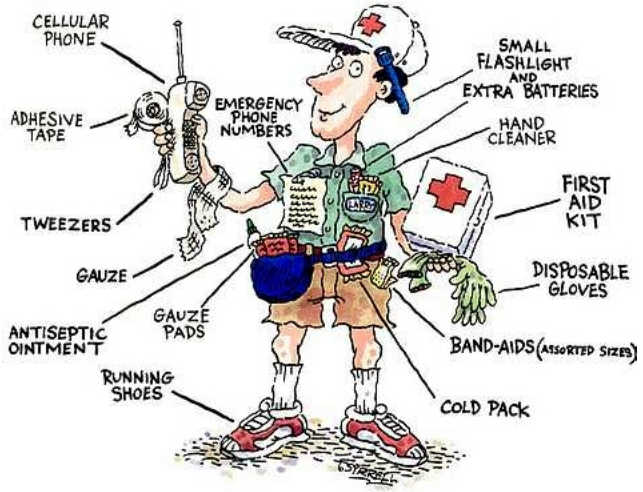
1. Training
2. Requirements
3. Design
4. Implementation
5. Verification
6. **Release**
7. Response

Public pre-release review

1. **Privacy**
2. Security



Planning



Preparation for incident response

Release

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

Final security and privacy review

Outcomes:

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

Release to manufacturing/release to web

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

Response

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

Execute Incident Response Plan

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

Process Models: wrapup

Microsoft SDL:

- Mature, long-term practical experience
- Heavyweight, ISV flavour
- Several supporting tools and methods

Other process models exist, with their pro's and con's

In general, no process will fit your organization perfectly

Mix-and-Match + adaptation are necessary

Agenda

1. Motivation
2. Process Models
3. **Maturity Models**
4. Agile Development
5. Conclusion

Why Maturity Models ?

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 tailor to the organization

Guidance related to security activities must be prescriptive

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

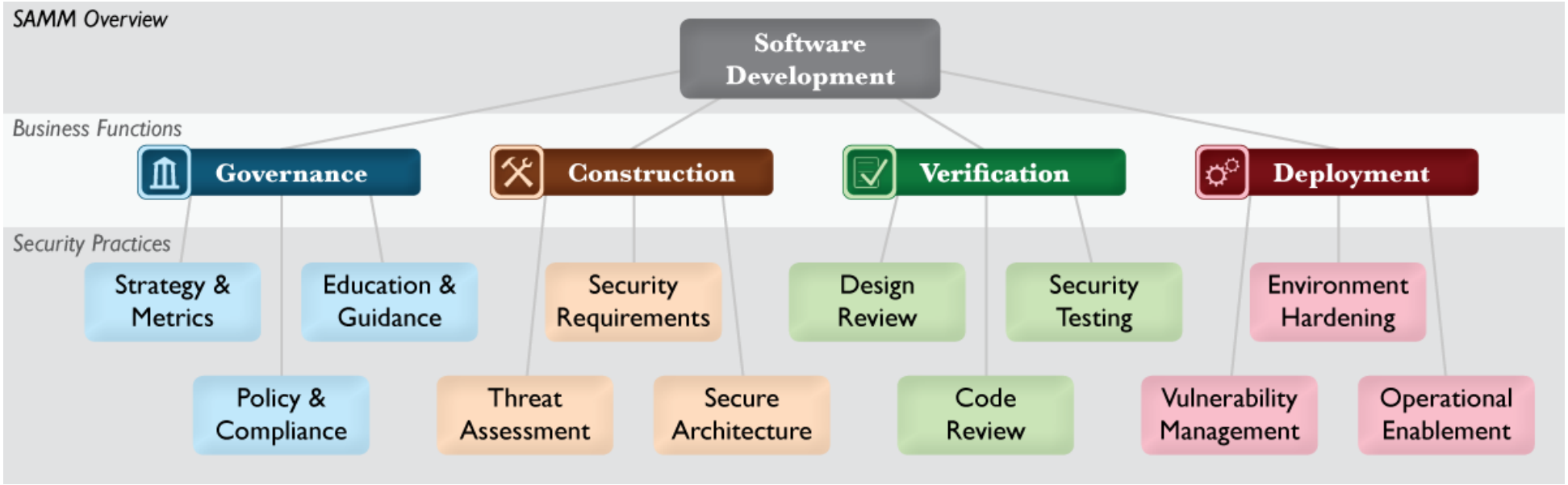
Overall, must be simple, well-defined, and measurable

Selected example: OpenSAMM

http://www.opensamm.org

Version 1.0, 2009

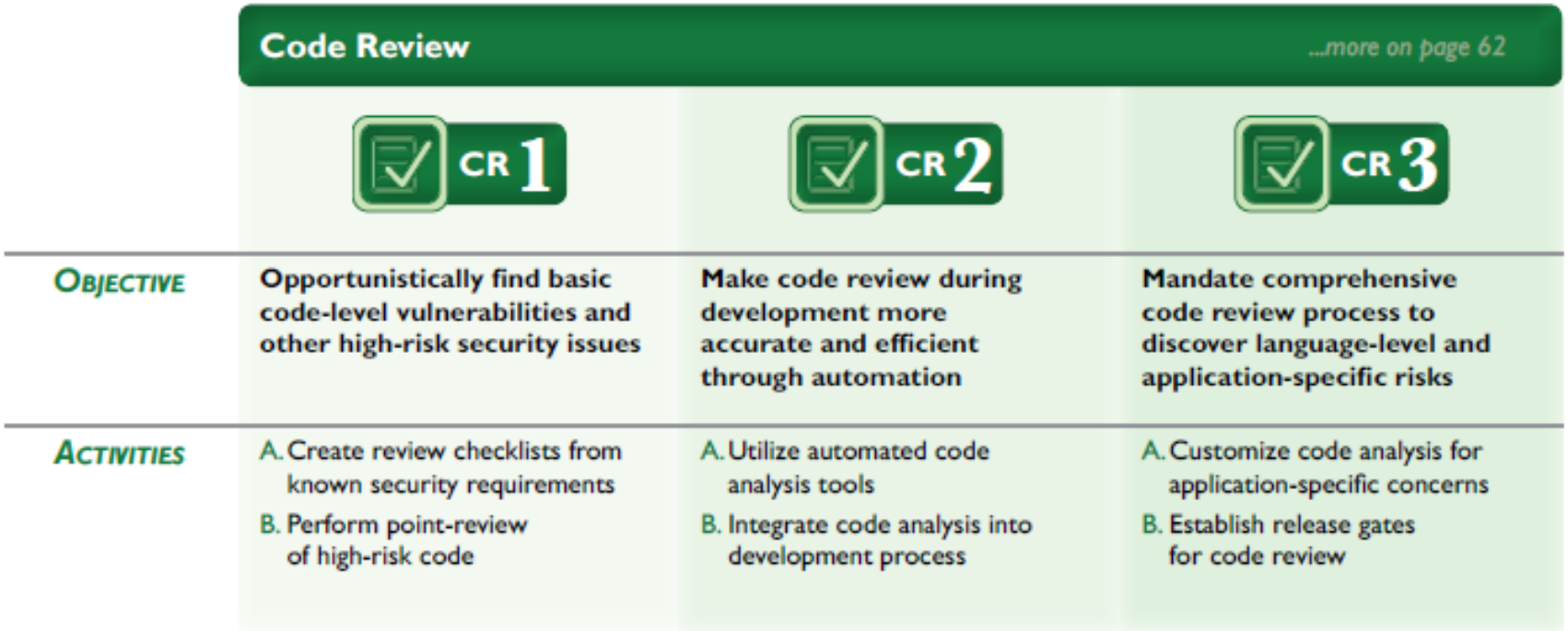
Core Structure



Notion of Maturity

Level	Interpretation
0	Implicit starting point representing the activities in the practice being unfulfilled
1	Initial understanding and ad-hoc provision of the security practice
2	Increase efficiency and/of effectiveness of the security practice
3	Comprehensive mastery of the security practice at scale

An example



Code Review			
	CR 1	CR 2	CR 3
OBJECTIVE	**Opportunistically find basic code-level vulnerabilities and other high-risk security issues**	**Make code review during development more accurate and efficient through automation**	**Mandate comprehensive code 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

...more on page 62

OpenSAMM also defines

- Objective
- Activities
- Results
- Success Metrics
- Costs
- Personnel
- Related Levels

Security Testing

ST 3

Require application-specific security testing to ensure baseline security before deployment

Activities

A. Employ application-specific security testing automation

Through either customization of security testing tools, enhancements to generic test case execution tools, or buildout of custom test harnesses, project teams should formally iterate through security requirements and build a set of automated checkers to test the security of the implemented business logic.

Additionally, many automated security testing tools can be greatly improved in accuracy and depth of coverage if they are customized to understand more detail about the specific software interfaces in the project under test. Further, organization-specific concerns from compliance or technical standards can be codified as a reusable, central test battery to make audit data collection and per-project management visibility simpler.

Project teams should focus on buildout of granular security test cases based on the business functionality of their software, and an organization-level team led by a security auditor should focus on specification of automated tests for compliance and internal standards.

B. Establish release gates for security testing

To prevent software from being released with easily found security bugs, a particular point in the software development life-cycle should be identified as a checkpoint where an established set of security test cases must pass in order to make a release from the project. This establishes a baseline for the kinds of security tests all projects are expected to pass.

Since adding too many test cases initially can result in an overhead cost bubble, begin by choosing one or two security issues and include a wide variety of test cases for each with the expectation that no project may pass if any test fails. Over time, this baseline should be improved by selecting additional security issues and adding a variety of corresponding test cases.

Generally, this security testing checkpoint should occur toward the end of the implementation or testing, but must occur before release.

For legacy systems or inactive projects, an exception process should be created to allow those projects to continue operations, but with an explicitly assigned timeframe for mitigation of findings. Exceptions should be limited to no more that 20% of all projects.

Results

- Organization-wide baseline for expected application performance against attacks
- Customized security test suites to improve accuracy of automated analysis
- Project teams aware of objective goals for attack resistance

Add'l Success Metrics

- >50% of projects using security testing customizations
- >75% of projects passing all security tests in past 6 months

Add'l Costs

- Buildout and maintenance of customizations to security testing automation
- Ongoing project overhead from security testing audit process
- Organization overhead from project delays caused by failed security testing audits

Add'l Personnel

- Architects (1 day/yr)
- Developers (1 day/yr)
- Security Auditors (1-2 days/yr)
- QA Testers (1-2 days/yr)
- Business Owners (1 day/yr)
- Managers (1 day/yr)

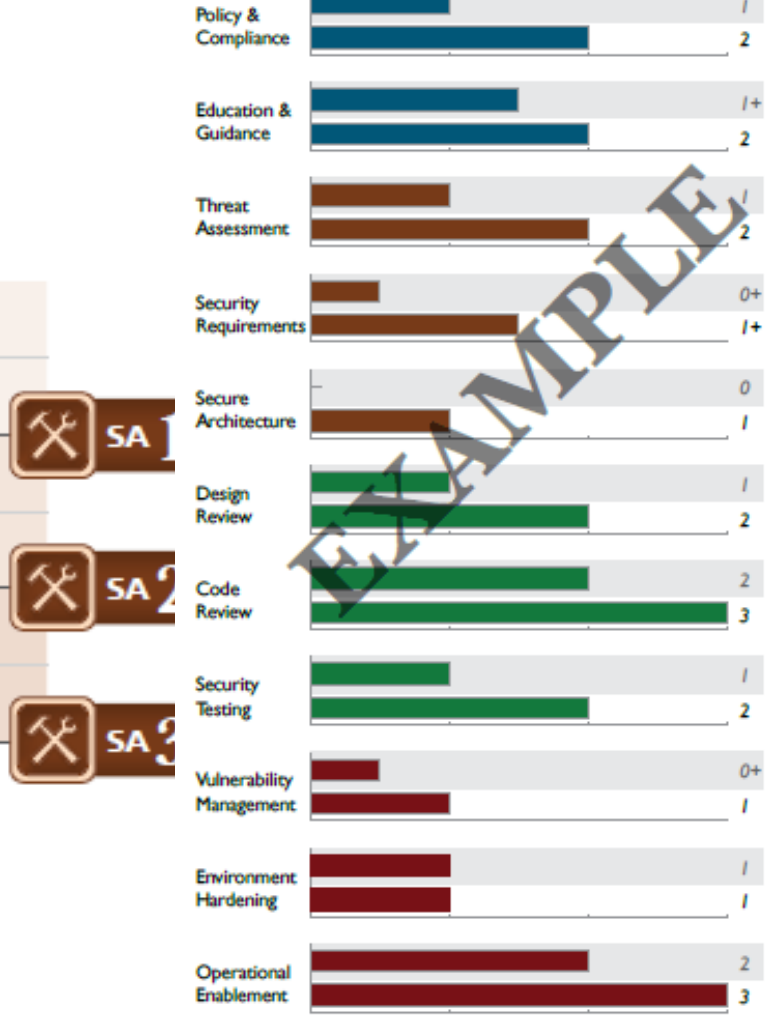
Related Levels

- Policy & Compliance - 2
- Secure Architecture - 3

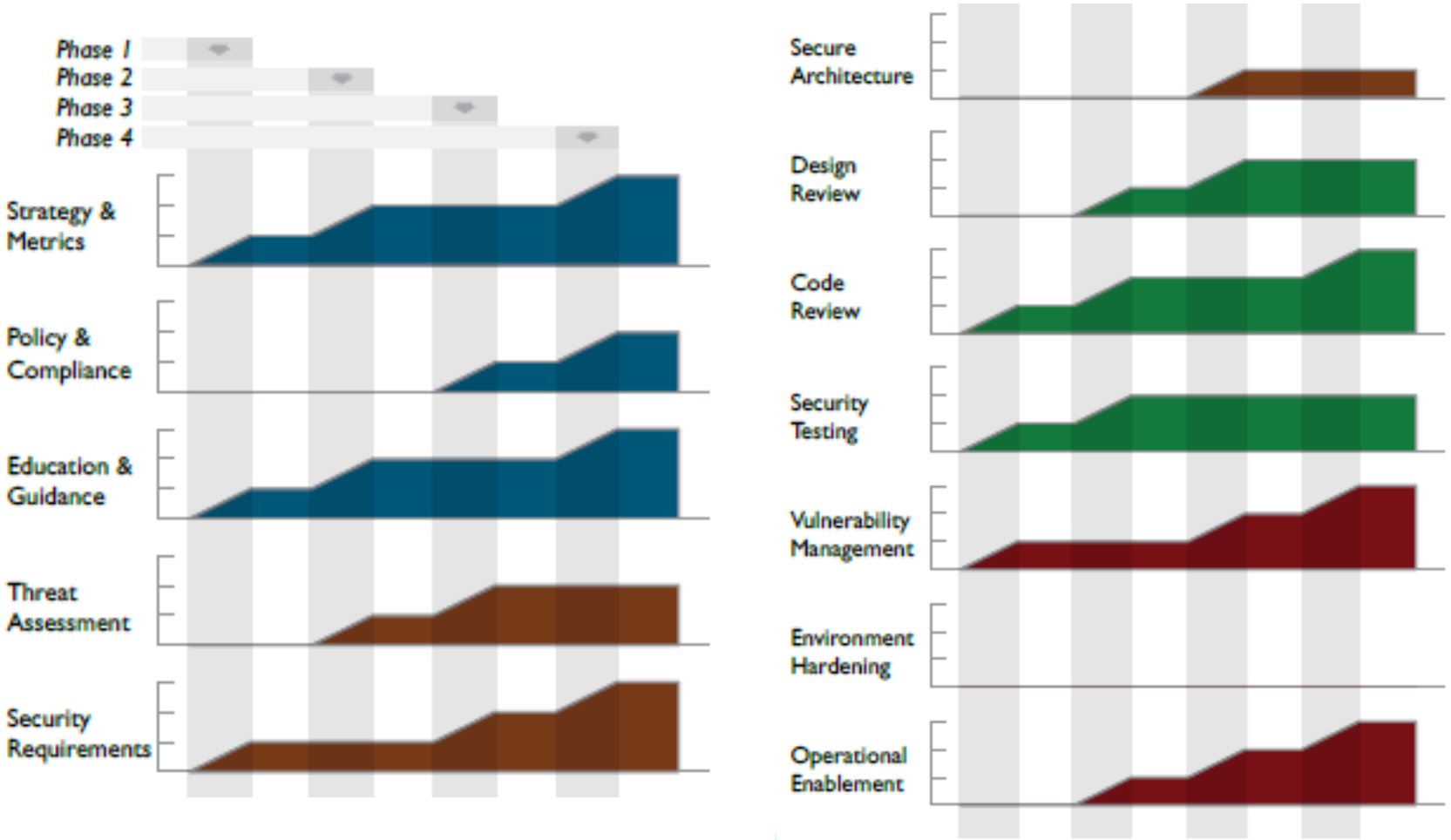
Practices - v1.0

Assessments

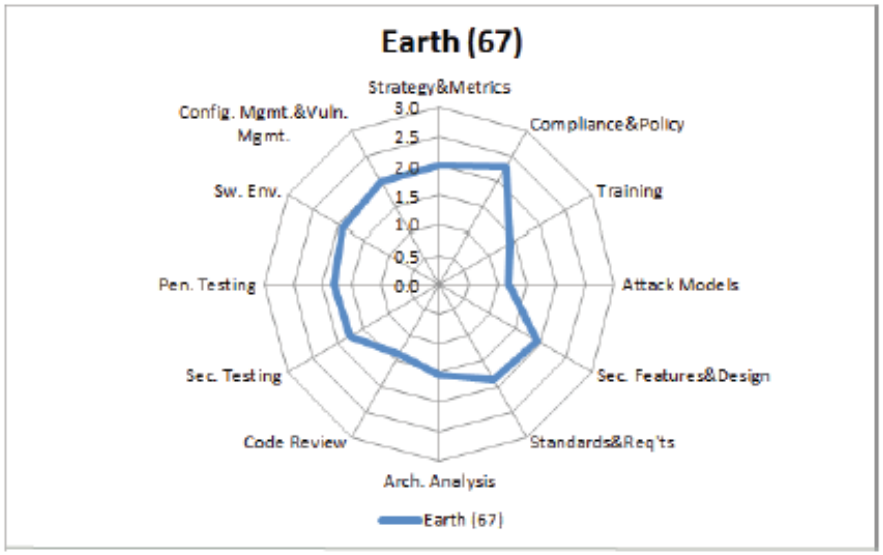
Secure Architecture	YES/NO
✦Are project teams provided with a list of recommended third-party components?	
✦Are most project teams aware of secure design principles and applying them?	
✦Do you advertise shared security services with guidance for project teams?	
✦Are project teams provided with prescriptive design patterns based on their application architecture?	
✦Are project teams building software from centrally controlled platforms and frameworks?	
✦Are project teams being audited for usage of secure architecture components?	

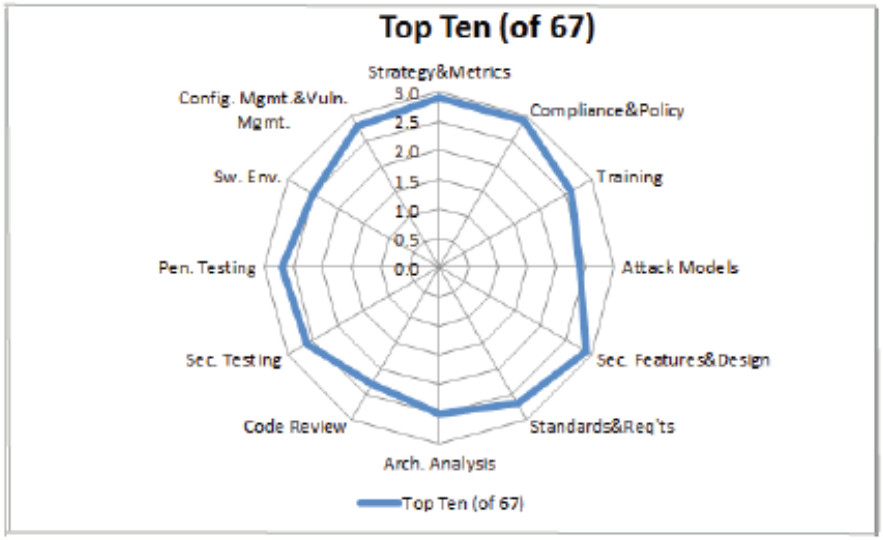


Roadmap templates per company type (ISV)



BSIMM5 statistics: summary





BSIMM5 statistics: per activity

Governance		Intelligence		SSDL Touchpoints		Deployment	
Activity	Observed	Activity	Observed	Activity	Observed	Activity	Observed
[SM1.1]	44	[AM1.1]	21	[AA1.1]	56	[PT1.1]	62
[SM1.2]	34	[AM1.2]	43	[AA1.2]	35	[PT1.2]	51
[SM1.3]	34	[AM1.3]	30	[AA1.3]	24	[PT1.3]	43
[SM1.4]	57	[AM1.4]	12	[AA1.4]	42	[PT2.2]	24
[SM1.6]	36	[AM1.5]	42	[AA2.1]	10	[PT2.3]	27
[SM2.1]	26	[AM1.6]	16	[AA2.2]	8	[PT3.1]	13
[SM2.2]	31	[AM2.1]	7	[AA2.3]	20	[PT3.2]	8
[SM2.3]	27	[AM2.2]	11	[AA3.1]	11		
[SM2.5]	20	[AM3.1]	4	[AA3.2]	4		
[SM3.1]	16	[AM3.2]	6				
[SM3.2]	6						
[CP1.1]	43	[SFD1.1]	54	[CR1.1]	24	[SE1.1]	34
[CP1.2]	52	[SFD1.2]	53	[CR1.2]	34	[SE1.2]	61
[CP1.3]	45	[SFD2.1]	26	[CR1.4]	50	[SE2.2]	31
[CP2.1]	24	[SFD2.2]	29	[CR1.5]	23	[SE2.4]	25
[CP2.2]	28	[SFD3.1]	9	[CR1.6]	25	[SE3.2]	10
[CP2.3]	29	[SFD3.2]	13	[CR2.2]	10	[SE3.3]	9
[CP2.4]	25	[SFD3.3]	9	[CR2.5]	15		
[CP2.5]	35			[CR2.6]	18		
[CP3.1]	14			[CR3.2]	4		
[CP3.2]	11			[CR3.3]	6		
[CP3.3]	8			[CR3.4]	1		
[T1.1]	50	[SR1.1]	48	[ST1.1]	51	[CMVM1.1]	59
[T1.5]	29	[SR1.2]	43	[ST1.3]	55	[CMVM1.2]	59
[T1.6]	23	[SR1.3]	45	[ST2.1]	27	[CMVM2.1]	50
[T1.7]	33	[SR1.4]	27	[ST2.4]	13	[CMVM2.2]	44
[T2.5]	9	[SR2.2]	23	[ST3.1]	11	[CMVM2.3]	30
[T2.6]	13	[SR2.3]	19	[ST3.2]	8	[CMVM3.1]	6
[T2.7]	9	[SR2.4]	19	[ST3.3]	6	[CMVM3.2]	6
[T3.1]	4	[SR2.5]	22	[ST3.4]	5	[CMVM3.3]	2
[T3.2]	4	[SR3.1]	8	[ST3.5]	7		
[T3.3]	8	[SR3.2]	12				
[T3.4]	9						
[T3.5]	5						

Maturity Models wrapup

OpenSAMM

- Comprehensive and rich model, more than just activities
- Supporting tools are available
- Real-world case studies, but few are openly shared

Other models exist with their pro's and con's

Maturity models provide an excellent framework for reasoning on software assurance, on a *strategic* level.

Agenda

1. Motivation
2. Process Models
3. Maturity Models
4. **Agile Development**
5. Conclusion

Agile Models: Rationale and Fundamentals

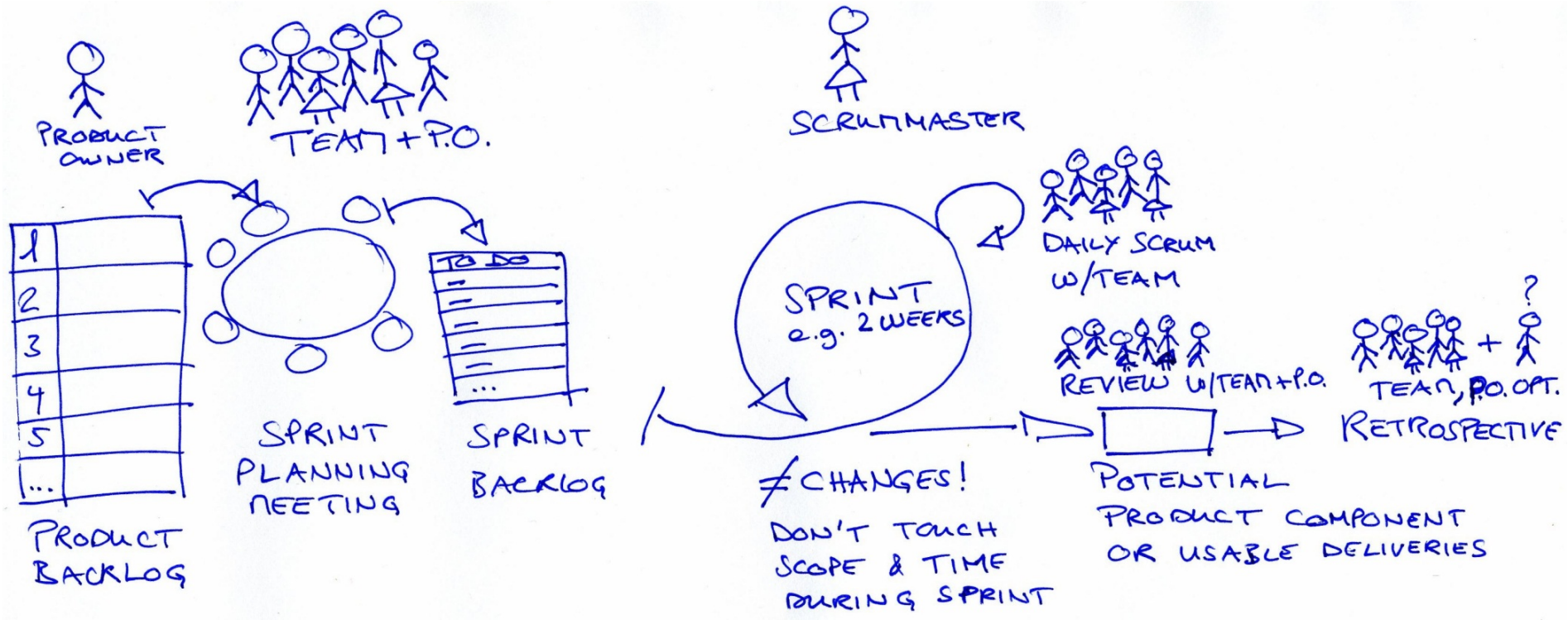
- Many traditional, large-scale software development projects are going wrong
  - Combination of business and technical causes
- Software is delivered late in the lifecycle
- Little flexibility during the process

Agile models focus on:

- Frequent interaction with stakeholders
- Short cycles

=> to increase flexibility and reduce risk

Agile Models: Scrum



Agile & Secure development: a mismatch ?

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

MS SDL-Agile

Basic approach: Fit SLD tasks to the backlog as non-functional stories

- Non-Technical vs. Technical
- Requirement vs. Recommendation

Each SDL task goes in one of three types of requirements:

Every-Sprint Requirements (excerpt)

- All team members must have had security training in the past year
- All database access via parameterized queries
- Fix security issues identified by static analysis
- Mitigate against Cross-Site Request Forgery
- Update Threat models for new features
- Use Secure cookies over HTTPS
- Link all code with the /nxcompat linker option
- Encrypt all secrets such as credentials, keys and passwords
- Conduct internal security design review

Bucket Requirements (excerpt)

Bucket A: Security Verification

- Perform fuzzing (network/ActiveX/File/RPC/…)
- Manual and automated code review for high-risk code
- Penetration testing

Bucket B: Design Review

- Conduct a privacy review
- Complete threat model training

Bucket C: Planning

- Define or update the security/privacy bug bar
- Define a BC/DR plan

One-Time Requirements (excerpt)

- Create a baseline threat model
- Establish a security response plan
- Identify your team's security expert
- Use latest compiler versions

Abuser Stories

Treat application security into software development by writing up application security risks as stories

- **Security stories**: “As a developer, I want to prevent SQLi into my application”
  - Not a real user story (not relevant for product owner, but to help the development team)
  - Never really finished
  - Cfr MS examples
- **Thinking like the bad guy**: “User X should not have access to this type of data”
  - Think about what users don’t want to and can’t do, how to trust users, what data is involved, …

Thou shall use Iteration Zero

Many agile projects start with an “Iteration Zero” to

- Get the team together
- Choose tools and frameworks
- Get to know the domain

This is an opportunity for security too, to

- Assign security responsibles
- Select security tools
- Determine risk levels

Security Involvement in the Process

Ensure that security-savvy people are involved at important phases:

- Planning game (to enhance/verify requirements)
- Development (daily follow-up)
- Review (to support acceptance)
- Retrospective (to improve dev. Practices for security)

Different profiles can be distinguished:

- Security architect
- Security engineer
- Risk Manager/Governance

Agenda

1. Motivation
2. Process Models
3. Maturity Models
4. Agile Development
5. **Conclusion**

Conclusions

SDLC is the framework for most of this week's sessions

No model is perfect, but they provide good guidance

Agile development can be improved as well

Take into account all cornerstones

Risk Management is key for rationalizing effort