www.pwc.com

SDLC Introduction and Process Models

SecAppDev 2017

Bart De Win

pwc

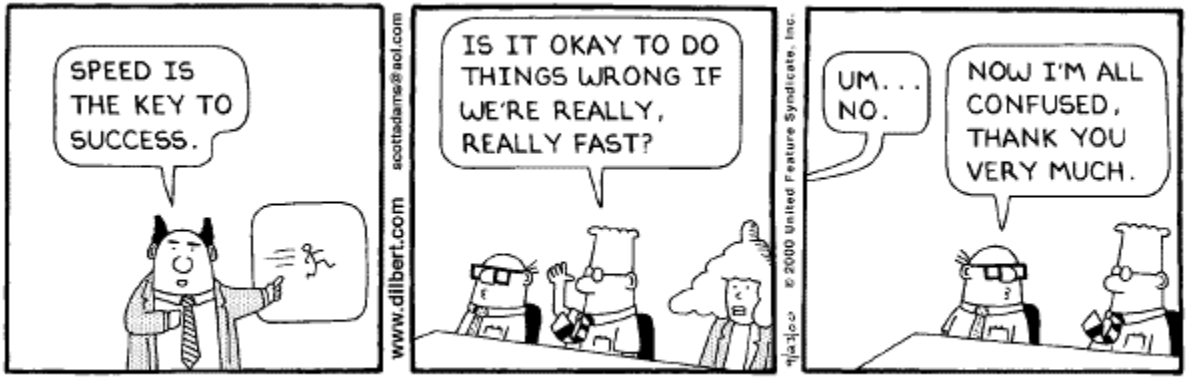
Bart De Win ?

- 20 years of Information Security Experience
  - Ph.D. in Computer Science - Application Security
- Author of >60 scientific publications
- ISC² CSSLP certified
- Senior Manager @ PwC Belgium:
  - Expertise Center Leader *Trusted Software*
  - (Web) Application tester (pentesting, arch. review, code review, …)
  - Proficiency in Secure Software Development Lifecycle (SDLC) and Software Quality
- OWASP SAMM co-leader
- Contact me at bart.de.win@be.pwc.com

Agenda

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

Application Security Problem



Software complexity

Technology stacks

Adaptability

Growing connectivity

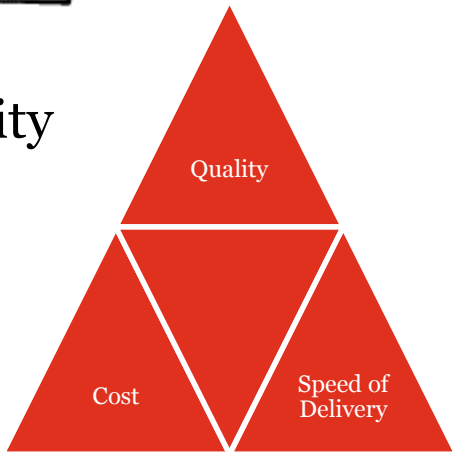
Better

Mobile

Cloud

Training

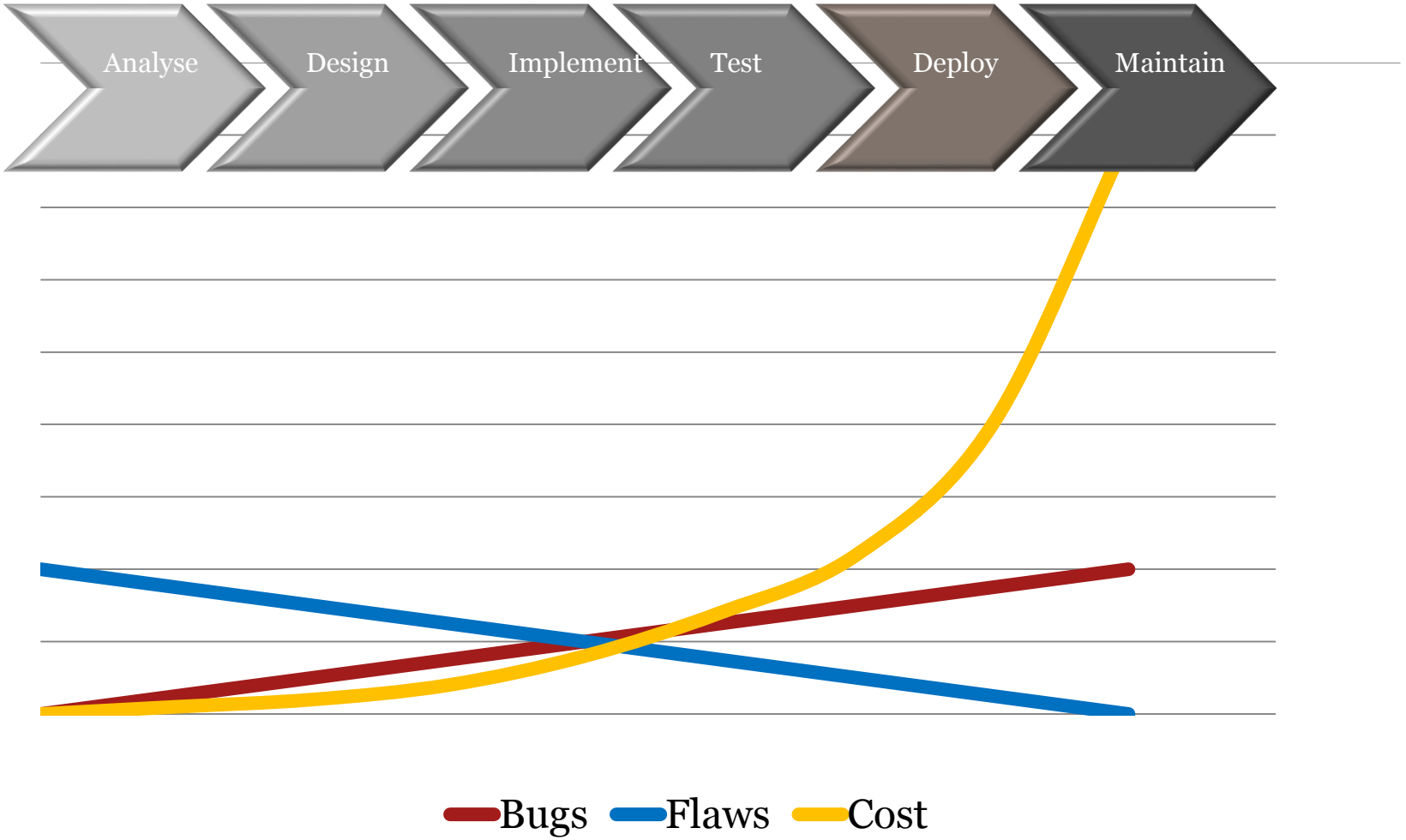
Faster



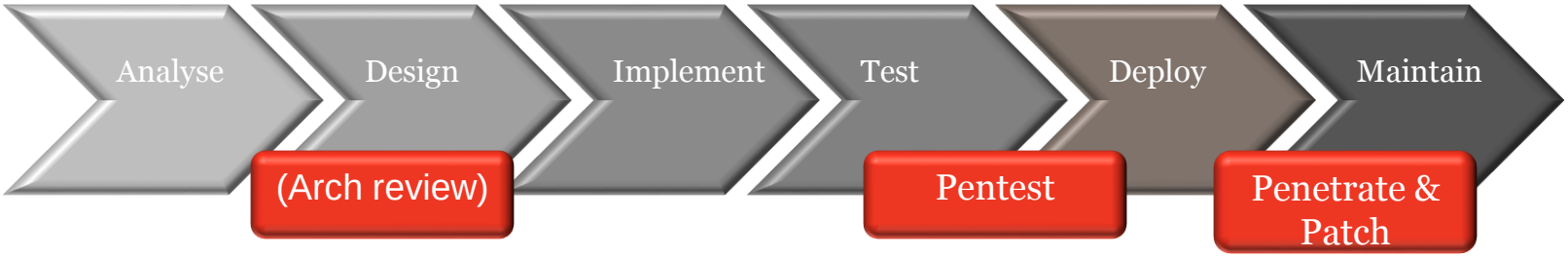
75% of vulnerabilities are application related

Application Security Symbiosis

Application Security during Software Development



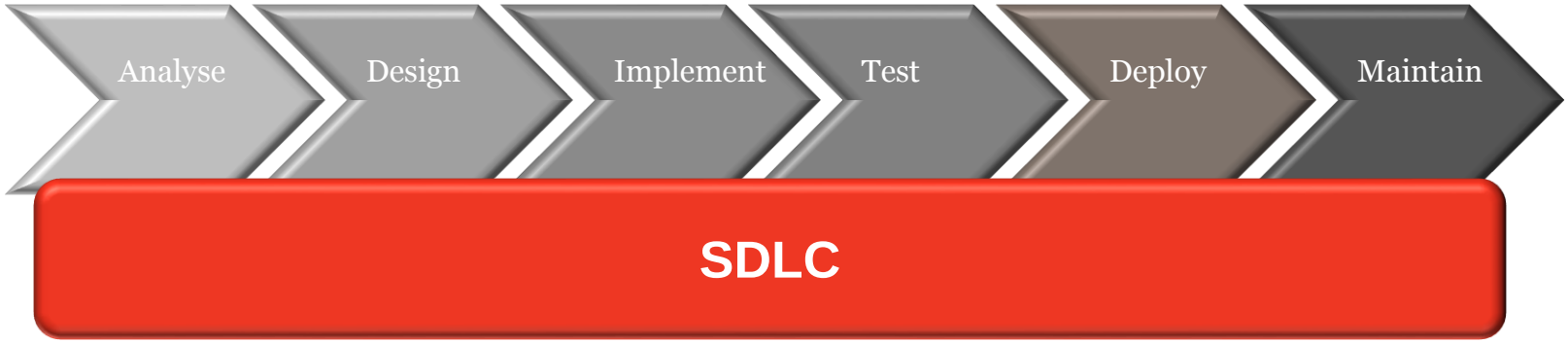
The State-of-Practice in Secure Software Development



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 ?

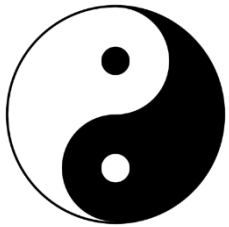


Enterprise-wide software security improvement program

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

SDLC Objectives & Principles

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



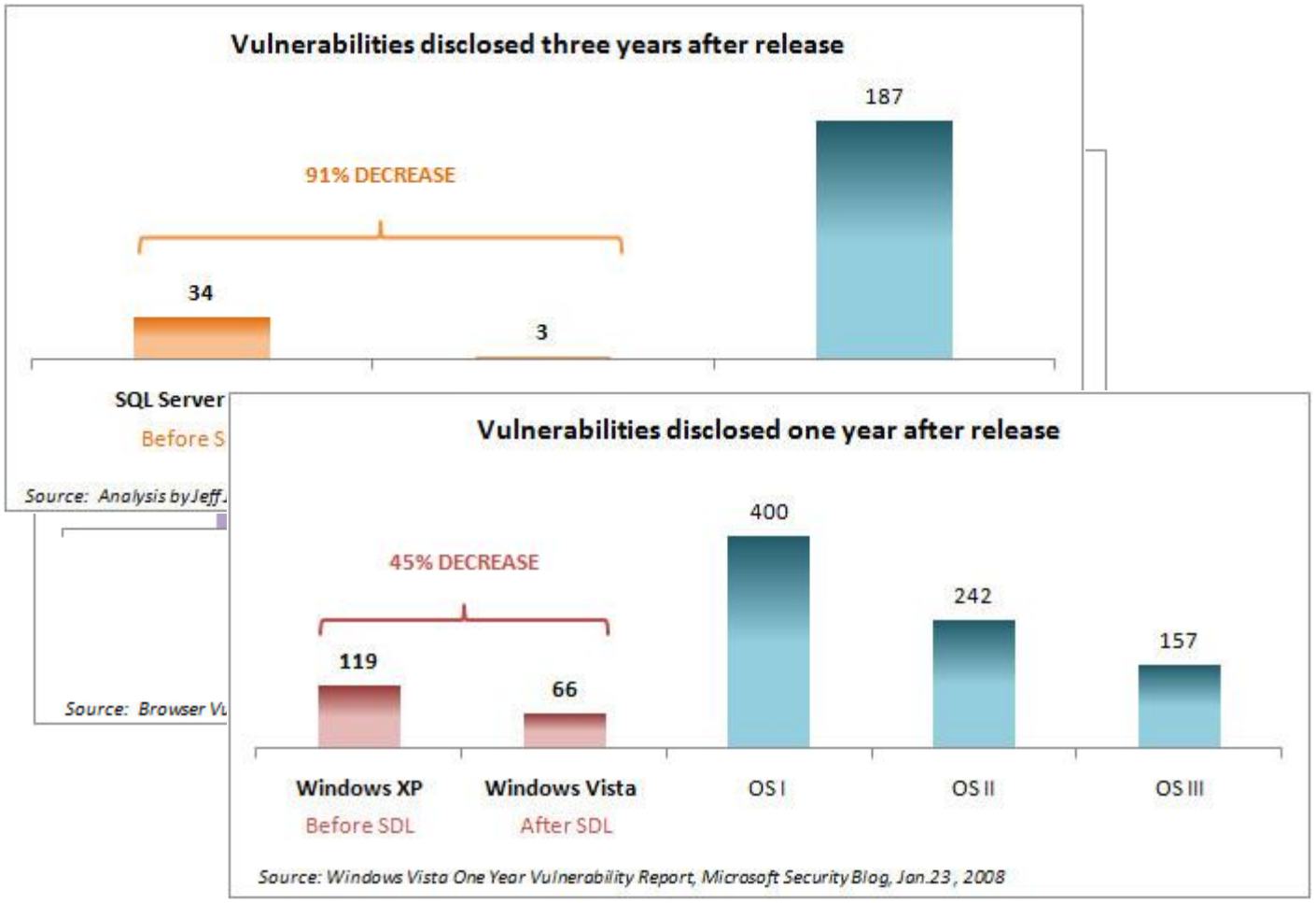

SDLC Cornerstones

Strategic ?

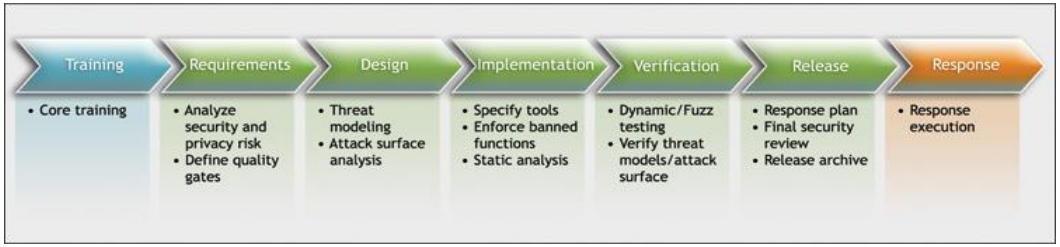
Organizations with a proper SDLC will experience an 80 percent decrease in critical vulnerabilities

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.

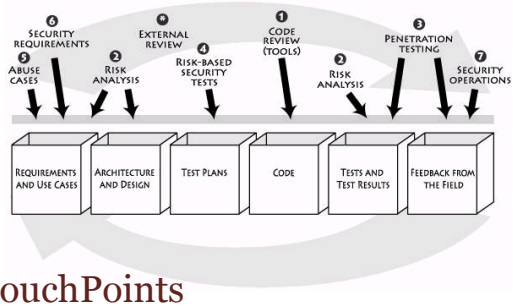
Does it really work ?



(Some) SDLC-related initiatives



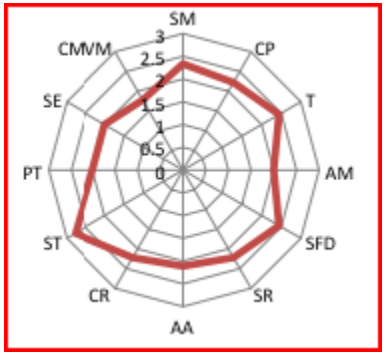
•Microsoft SDL



•TouchPoints

NIST National Institute of Standards and Technology

•SP800-64



•BSIMM

SSE CMM

•SSE-CMM

•CLASP

1000 1 SAFECode
011111
1000 1 Software Assurance Forum for Excellence in Code
111110 Driving Security and Integrity
1000 1

Gartner

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

•SAMM

Software Engineering Institute | Carnegie Mellon

•TSP-Secure

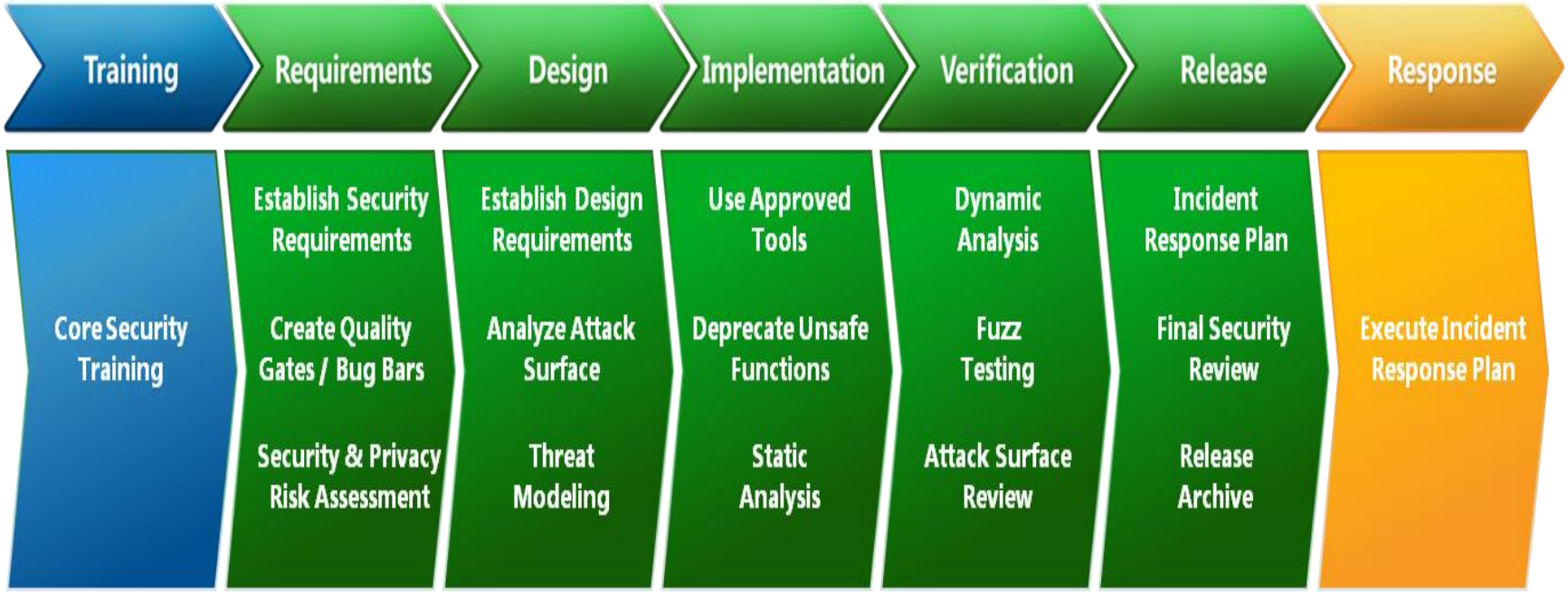
•GASSP

ISO/IEC 27034

Agenda

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

Selected Example: Microsoft SDL (SD3+C)



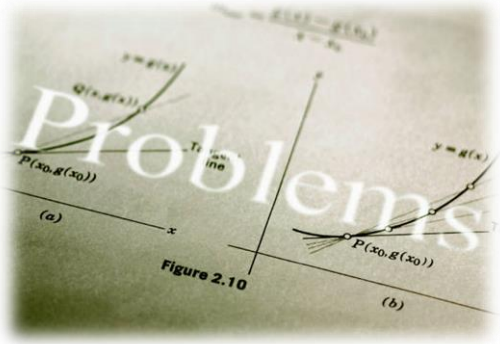
Training

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

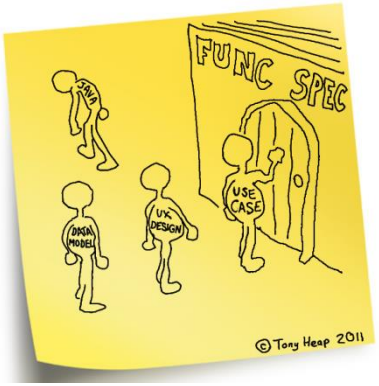
Cost analysis

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

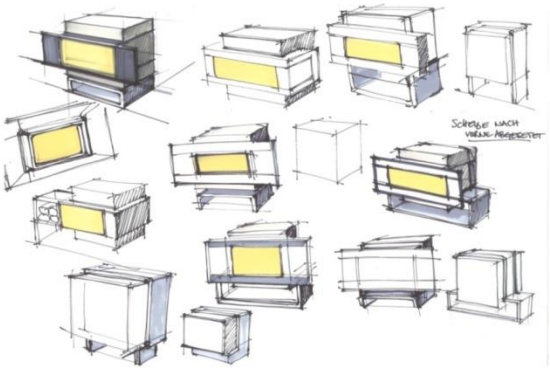
Design

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

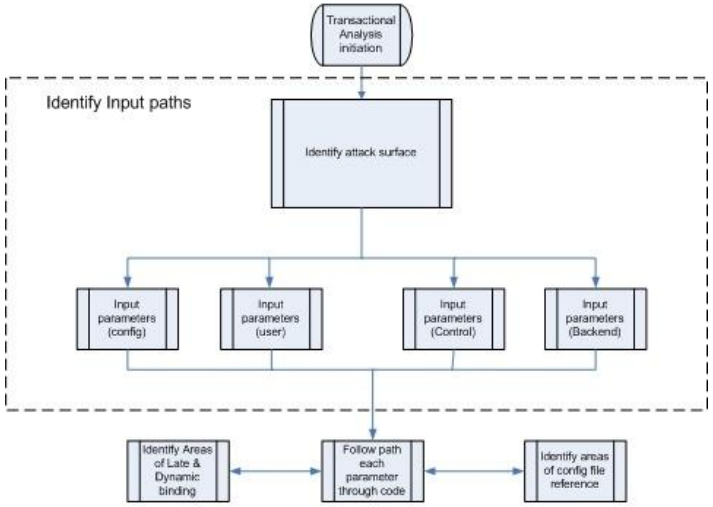
Establish and follow best practices for Design



≠ secure-coding best practices



Risk analysis



Threat modeling

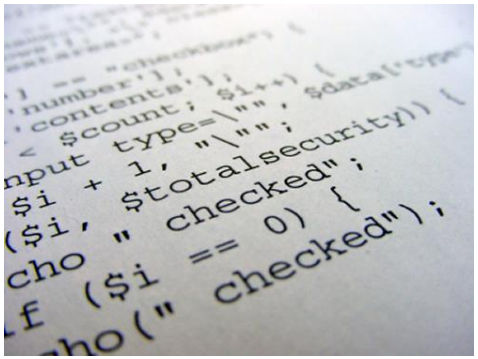
STRIDE

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

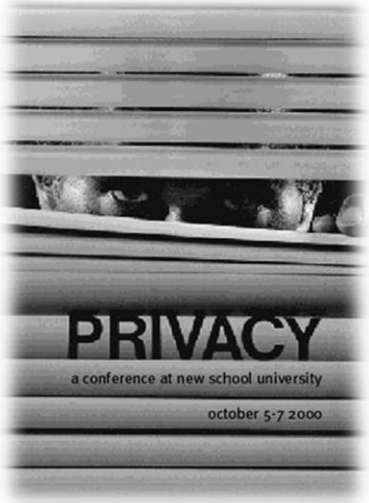


1. Review available information resources
2. Review recommended development tools
3. Define, communicate and document all best practices and policies

Verification

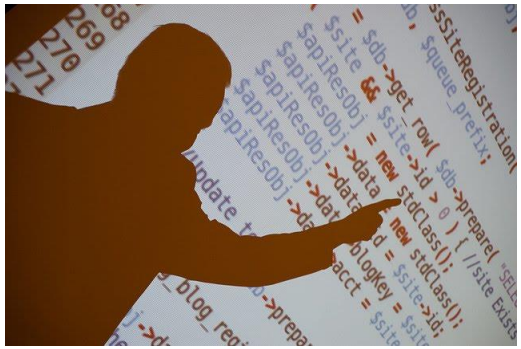
1. Training
2. Requirements
3. Design
4. Implementation
5. **Verification**
6. Release
7. Response

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

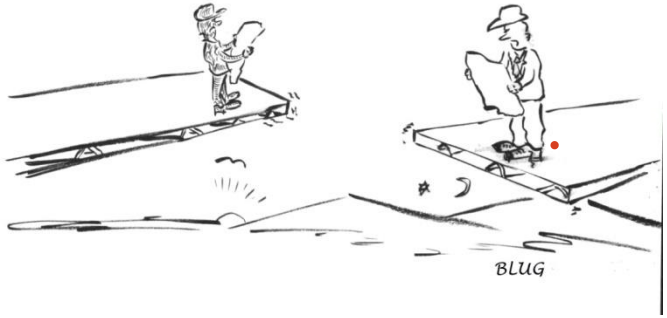


Release

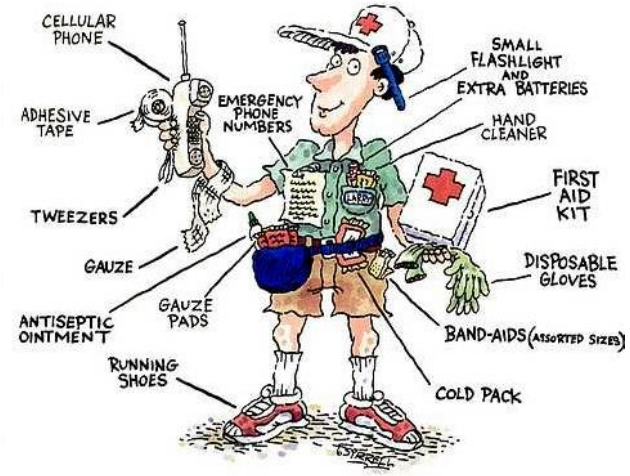
1. Training
2. Requirements
3. Design
4. Implementation
5. Verification
6. **Release**
7. Response

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. **Agile Development**
4. Good Practices
5. Conclusion

Agile Models: Rationale and Fundamentals

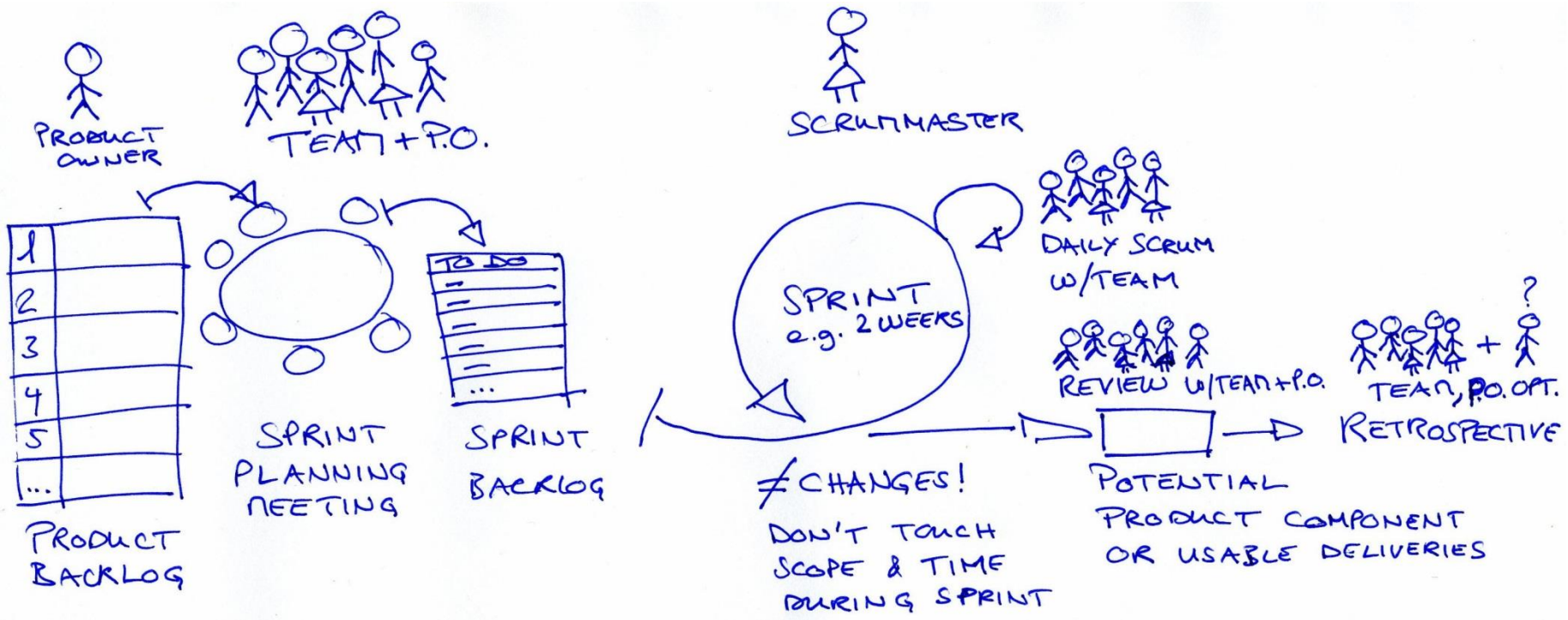
- Many traditional, large-scale software development projects are going wrong
  - Combination of business and technical causes
- Software is delivered late in the lifecycle
- Little flexibility during the process

Agile models focus on:

- Frequent interaction with stakeholders
- Short cycles

=> to increase flexibility and reduce risk

Agile Models: Scrum



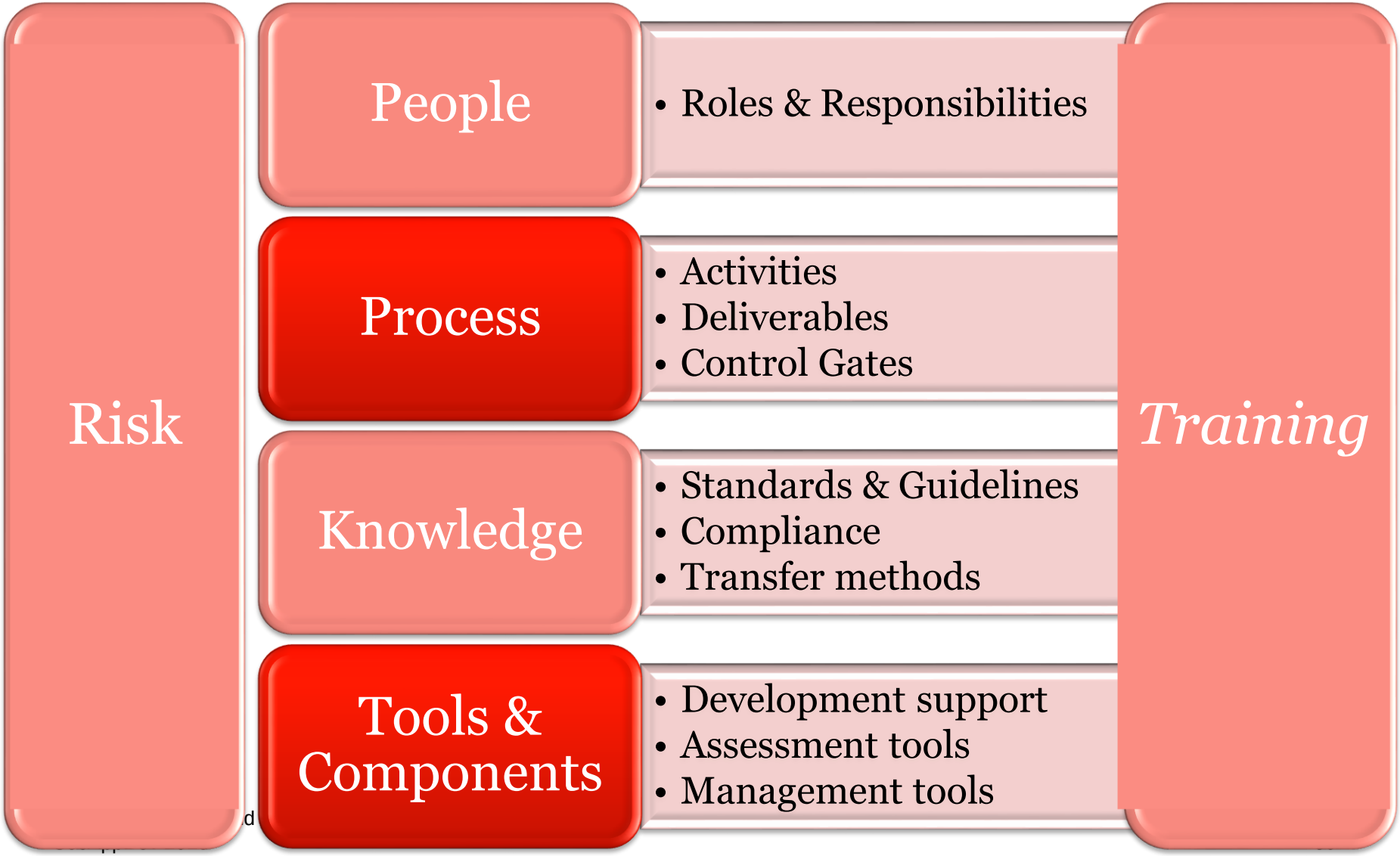
Agile & Secure development: a mismatch ?

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

Secure Agile is …

enablement, rather than control

Secure Agile – Where's the difference ?



Secure Agile: general principles

- Make security a natural part of the process, but don't overdo
  - lightweight, in-phase and iterative
  - preventative and detective controls
- Be involved at key moments in the process
- Leverage important agile concepts
- Small steps at a time (i.e. continuous improvement)

User Stories

- Capture security requirements, policies and regulations in user stories
- Simple, concrete and actionable
- Reusable?

"As a <type of user>,
I want <some goal>
so that <some reason>."

- Mark all user stories with security labels

- Integrate security into user stories as:
  - Definition of Done
  - Acceptance criteria

Threat Modelling & Abuser Stories

Consider writing 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
- **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, …

Sprint Planning

- Features to be implemented per sprint are selected during sprint planning.

- Ensure security tasks are not “stuck” on the backlog
  - Presence of security-savvy person during sprint planning
  - Establish rules to deal with security stories
  - Security labels can be used to drive selection

Example: MS SDL-Agile

Basic approach: Fit SDL 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	Bucket	One-Time

Example: 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

Example: 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

Example: One-Time Requirements (excerpt)

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

Security testing

- Automated testing is an important element in agile quality control

- For security, this can be realized by:
  - Unit testing (e.g., authorisation checks, logging, …)
  - Regression testing
  - Static analysis (SAST) based on coding guidelines
  - Dynamic analysis (DAST) based on scenarios and/or vulnerability tests
  - Fuzzing

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

Secure Agile process: key take-aways

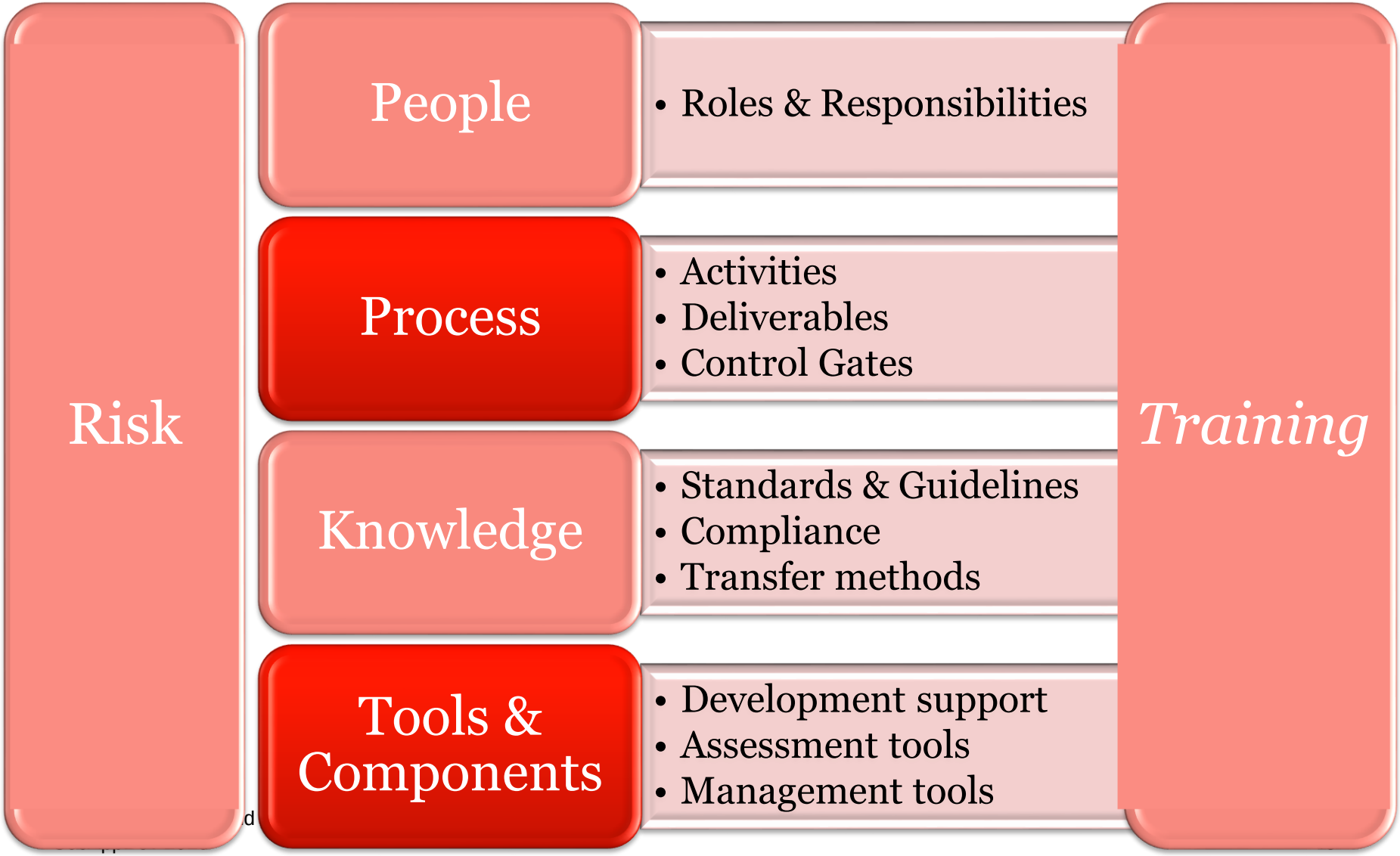
Ensure that security-savvy people are involved at important phases:

- 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

Secure Agile – Where's the difference ?



Secure Agile Tool Chain: general principles

- Secure agile is about enabling, rather than controlling
  - Embedding security tools to support development
- Given short sprint cycles, automation is important.

- Good tools do:
  - Work continuously (to avoid developers being blocked)
  - Integrate well into developer's world
  - Avoid causing too much overhead or confusion

- Evaluate carefully which tools to implement (and which to avoid)

Secure Coding

Integrate security tools in the development IDE's:

- Support for secure coding guidelines
- Static analysis tools

Ensure common development environment:

- Programming run-time
- Security components (e.g., SSO IdP's, …)

Proper source control and versioning

Security testing

Daily	Per sprint	Before release
• Unit tests • Regression tests • Peer reviews	• Static Analysis • Dynamic Analysis • Fuzzing	• Penetration testing

- Integrated with backlogs where appropriate

Secure Build

Central build, using central configuration files

Consider:

- Code signing
- Obfuscation
- …

Secure Deploy / DevOps

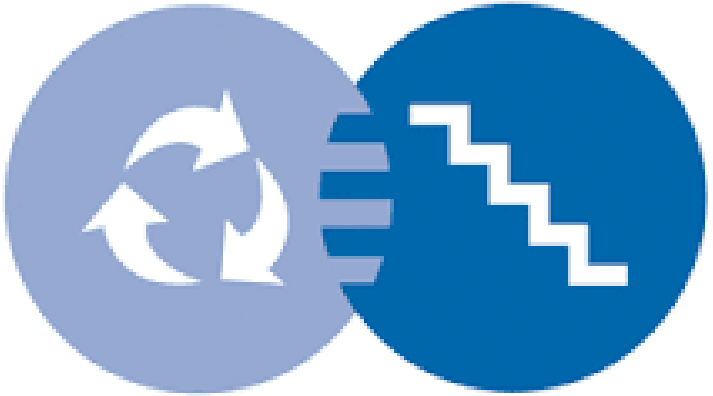
- Automated deploy, using central configuration files

- Consider:
  - Random key generation
  - Appropriate key/certificate protection (config files, key stores, …)
  - Proper hardening of application servers
  - Security appliance configuration (e.g., WAF)
  - Security monitoring
  - …

Hybrid models

- Many companies are combining waterfall and agile
  - Studies indicate better resulting quality

- For security, easier to hook into
  - E.g., full architecture cycle



Best Practices / Lessons Learned

- Use small steps at a time – the agile way
- Build on agile concepts (backlog, retrospective)
  - Find a way to prioritize security in the planning
- Use automation as much as possible
- Review samples independent of project sprints
- Rely on security champions
  - E.g., security requirements, design review, code review
- Agile should not be an excuse for not having documentation

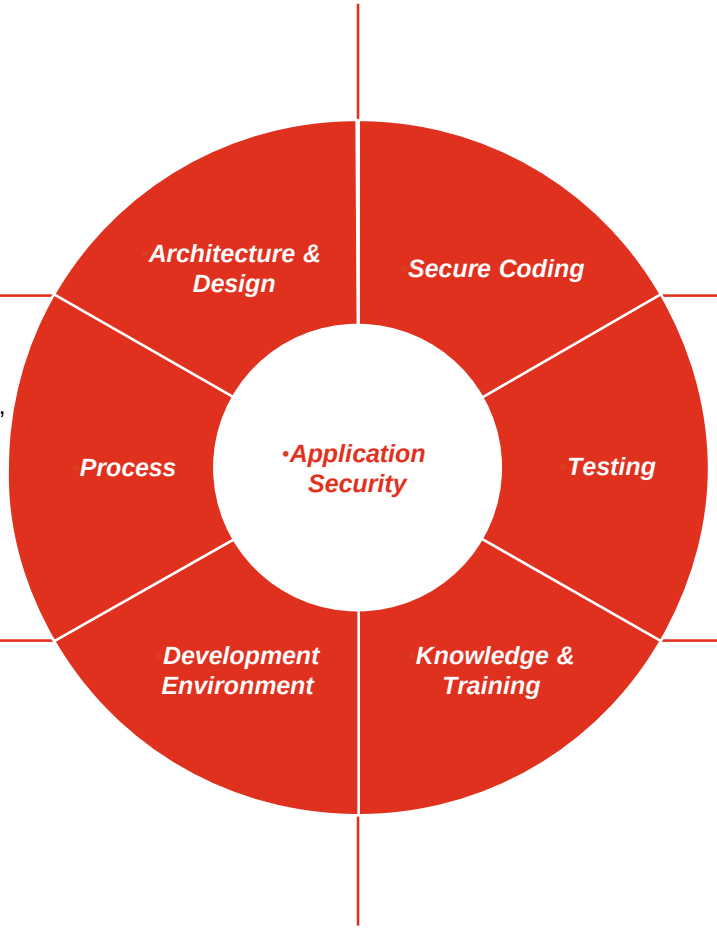
Agenda

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

Good Practices

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

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



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

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

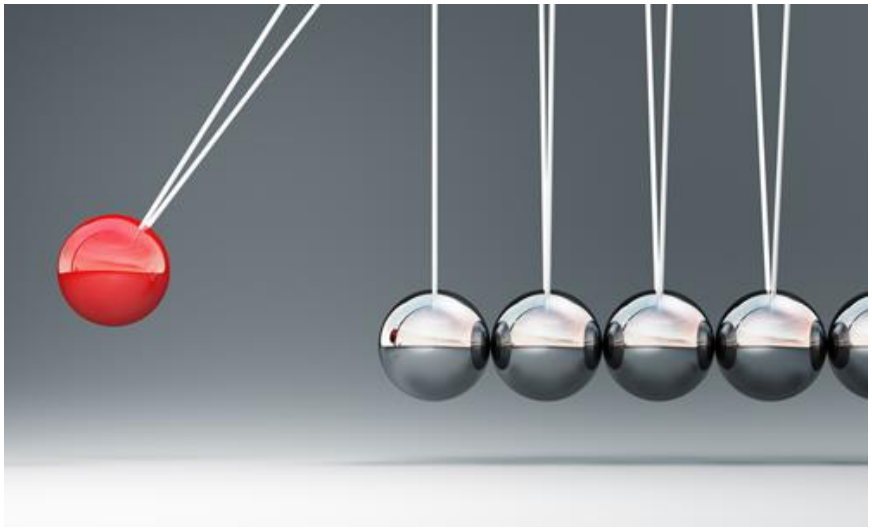
- Standardized dev. environment
- Central code repository
- Central build system
- Controlled promotion mechanism
- Continuous integration
- Screen & scan external libraries
- Regularly update tools & libraries
- Host third-party libraries locally

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

SDLC impact

Difficult to predict, but:

- Projects are estimated to increase with 5 – 15% for security
- ROI is achievable taking maintenance and incident management into account
- SDLC capability costs approx. 1 FTE/100 developers



Agenda

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

Conclusions

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

No model is perfect, but they provide good guidance

Take into account all cornerstones

Risk Management is key for rationalizing effort

SDLC Cornerstones