# Software security lifecycle

Nuno Silva, PhD, Critical Software SA

E.: npsilva@ua.pt; M: 932574030

**Mestrado em Cibersegurança – Robust Software**

# Agenda

- Motivation
- Objectives
- Secure SW Lifecycle Processes
- Secure SW Lifecycle Processes Summary
- Assessing the Secure Software Lifecycle
- Benefits
- References

# Motivation

- One intrinsic motivation to security is “self-improvement”, where the developer challenges one-self to write secure code. “Sometimes I will have the challenge, that ‘okay, this time I’m going to submit [my code] for a review where nobody will give me a comment’.”
- Professional responsibility and concern for users are two extrinsic motivations, where the action is not performed for its inherent enjoyment, but rather to fulfill what the developer views as their responsibility to their profession and to safeguard users’ privacy and security. “I would not feel comfortable with basically having something used by end users that I didn’t feel was secure, or I didn’t feel respective of privacy, umm so I would try very hard to not compromise on that.”

# Motivation

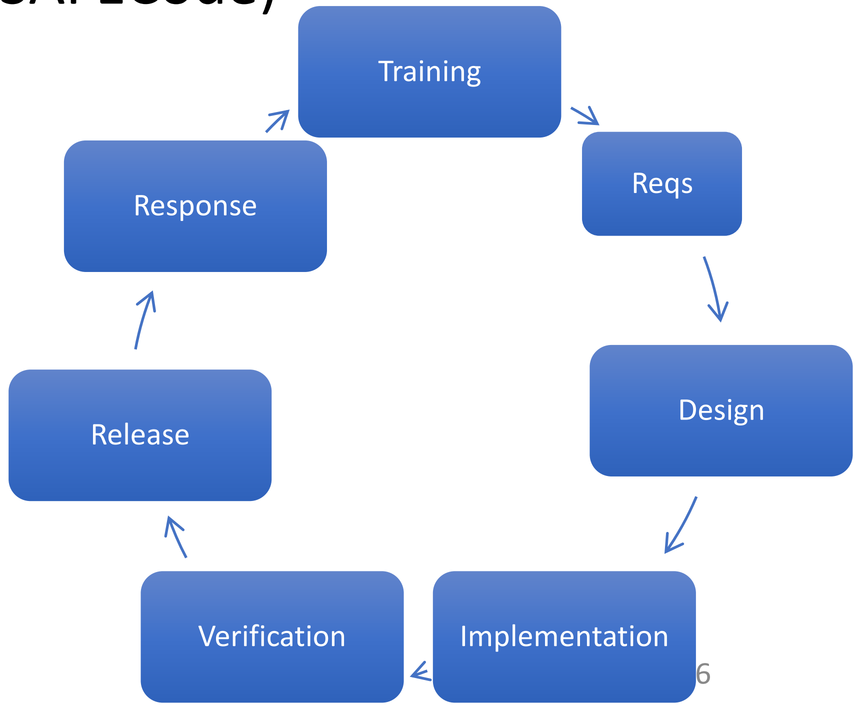
- Lack of resources and the lack of support are two factors that led to a perceived lack of competence to address software security. “We don’t have that much manpower to explicitly test security vulnerabilities, [..] we don’t have those kind of resources. But ideally if we did have [a big] company, I would have a team dedicated to find exploits. But unfortunately we don’t.”
- Lack of interest, relevance, or value of performing security tasks. The lack of relevance could happen when security is not considered one of the developer’s everyday duties (not my responsibility), or when security is viewed as another entity’s responsibility (security is handled elsewhere), such as another team or team-member. “I don’t really trust [my team members] to run any kind of, like, source code scanners or anything like that. I know I’m certainly not going to.”
- Ref: Motivations and Amotivations for Software Security, Halla Assal, Sonia Chiasson, Carleton Univ., Canada, https://wsiw2018.l3s.uni-hannover.de/papers/wsiw2018-Assal.pdf, visited: 12-10-2020.

# Objectives

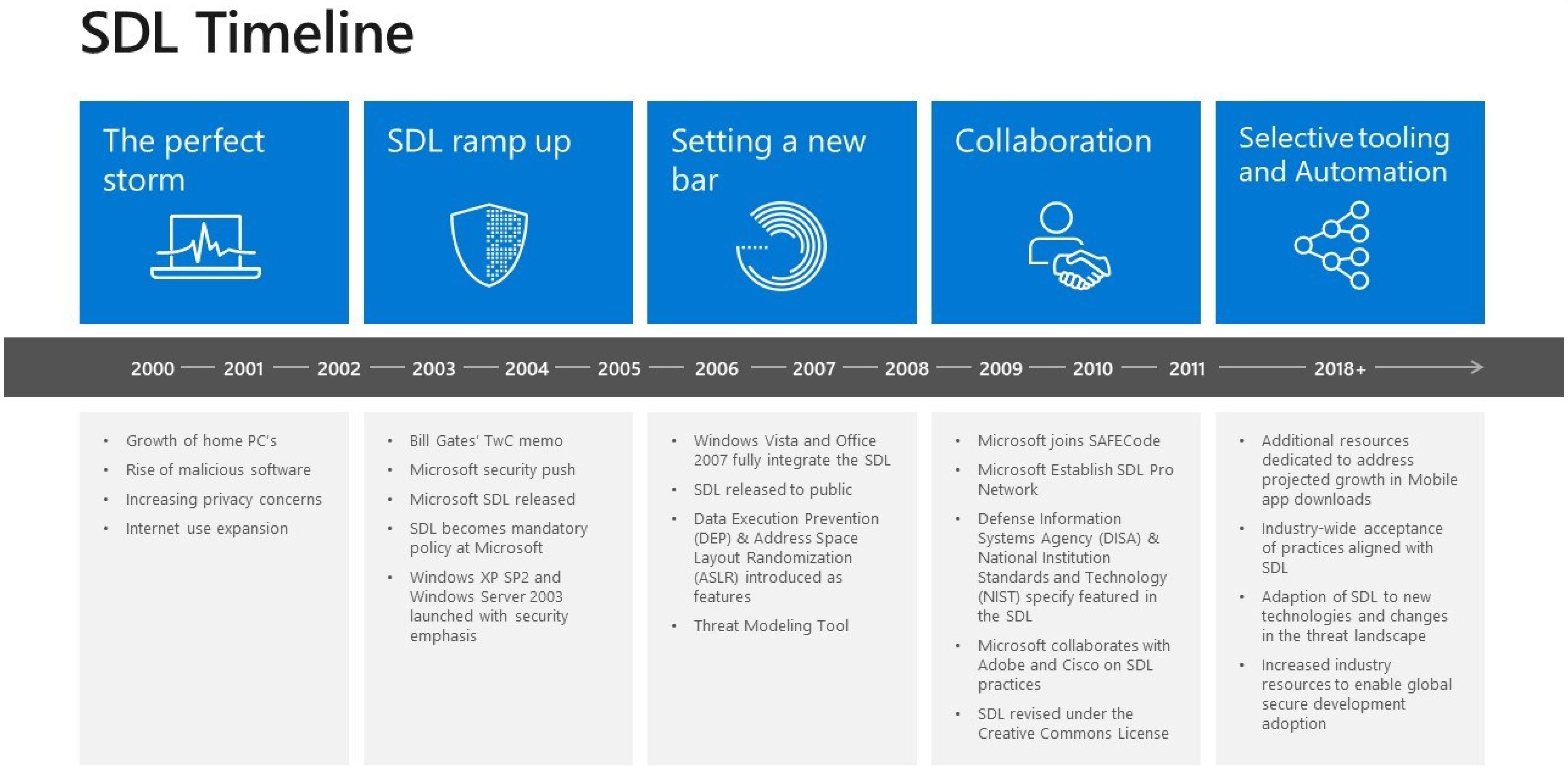
- Consider security concerns as early as possible in the development process.
- Be aware of trends in software change management
- Control application security management
- Be able to apply the security life cycle
- Make security part of SW Engineering, part of the culture.

# Secure SW Lifecycle Processes

- Three “Good” Examples:
  - Microsoft Security Development Lifecycle (SDL)
  - Software Security Touchpoints
  - Software Assurance Forum for Excellence in Code (SAFECode)



# SDL



## SDL Timeline

2000 — 2001 — 2002 — 2003 — 2004 — 2005 — 2006 — 2007 — 2008 — 2009 — 2010 — 2011 —— 2018+ ——→

### The perfect storm

- Growth of home PC's
- Rise of malicious software
- Increasing privacy concerns
- Internet use expansion

### SDL ramp up

- Bill Gates' TwC memo
- Microsoft security push
- Microsoft SDL released
- SDL becomes mandatory policy at Microsoft
- Windows XP SP2 and Windows Server 2003 launched with security emphasis

### Setting a new bar

- Windows Vista and Office 2007 fully integrate the SDL
- SDL released to public
- Data Execution Prevention (DEP) & Address Space Layout Randomization (ASLR) introduced as features
- Threat Modeling Tool

### Collaboration

- Microsoft joins SAFECode
- Microsoft Establish SDL Pro Network
- Defense Information Systems Agency (DISA) & National Institution Standards and Technology (NIST) specify featured in the SDL
- Microsoft collaborates with Adobe and Cisco on SDL practices
- SDL revised under the Creative Commons License

### Selective tooling and Automation

- Additional resources dedicated to address projected growth in Mobile app downloads
- Industry-wide acceptance of practices aligned with SDL
- Adaption of SDL to new technologies and changes in the threat landscape
- Increased industry resources to enable global secure development adoption

Ref: https://www.microsoft.com/en-us/securityengineering/sdl

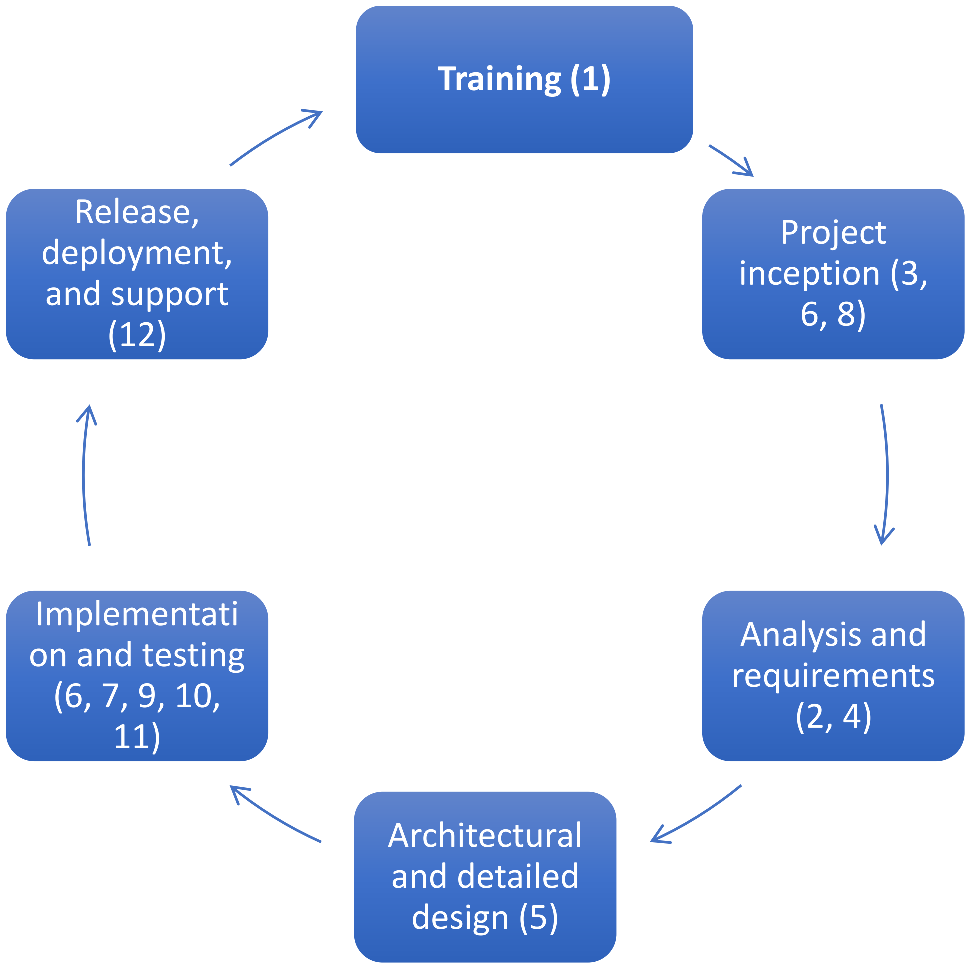
# SDL

1. Cyber security related training
2. Elicitation of explicit Security Requirements
3. Define Metrics, Report Compliance
4. Apply Threat Modelling (security risks analysis)
5. Establish Design Requirements
6. Define and use Cryptography Standards

# SDL

7. Manage Security Risk when Using 3rd 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

# SDL::Overview



Training (1)

Project inception (3, 6, 8)

Analysis and requirements (2, 4)

Architectural and detailed design (5)

Implementation and testing (6, 7, 9, 10, 11)

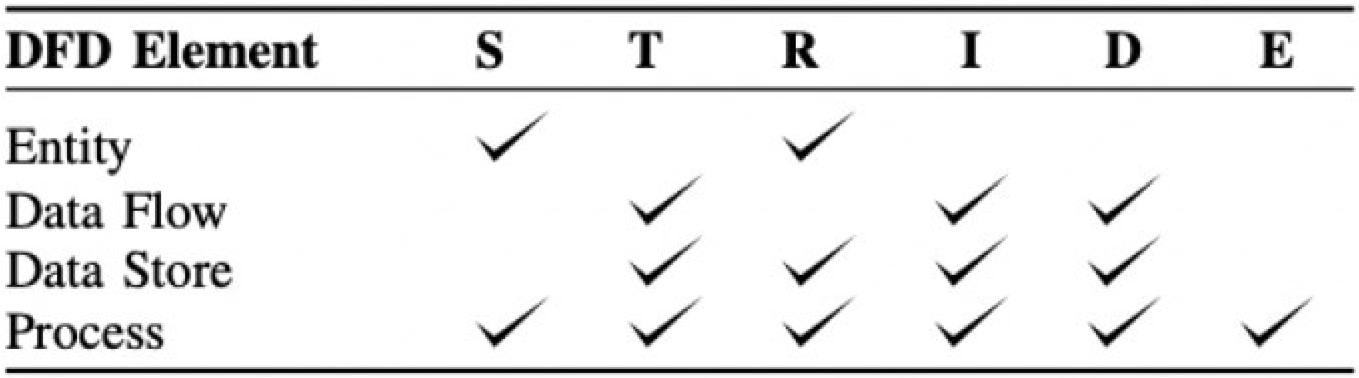
Release, deployment, and support (12)

# SDL::Apply Threat Modelling

- STRIDE approach
  - Spoofing Identity (pretend to be someone else)
  - Tampering with data (malicious modification of data)
  - Repudiation (denying performance of an action)
  - Information Disclosure (exposure of classified info)
  - Denial of Service (service unavailable or unusable)
  - Elevation of priviledge (access to more elevated priviledges)
- STRIDE, Atack trees, Architectural Risk Analysis help in enumerating threats and act upon the design to eliminate or control the impacts.

# SDL::Apply Threat Modelling

- 1. List Assets / Components;
- 2. Identify Threats (e.g. with data flow diagrams);
- 3. Identify and Classify (STRIDE) the vulnerabilities;



| DFD Element | S | T | R | I | D | E |
|---|---|---|---|---|---|---|
| Entity | ✓ |  | ✓ |  |  |  |
| Data Flow |  | ✓ |  | ✓ | ✓ |  |
| Data Store |  | ✓ | ✓ | ✓ | ✓ |  |
| Process | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

- Ref: DFD element mapping to the STRIDE Framework (Khan, Rafiullah & Mclaughlin, Kieran & Laverty, David & Sezer, Sakir., 2017)

# SDL::Apply Threat Modelling

- More reading on STRIDE:
  - Mahmood, Haider, (2017). *Application Threat Modeling using DREAD and STRIDE.* Accessed 12-10-2020 at [https://haiderm.com/application-threat-modeling-using-dread-and-stride](https://haiderm.com/application-threat-modeling-using-dread-and-stride).
  - Khan, R., McLaughlin, K., Laverty, D., & Sezer, S. (2018). STRIDE-based Threat Modeling for Cyber-Physical Systems. In 2017 IEEE PES: Innovative Smart Grid Technologies Conference Europe (ISGT-Europe): Proceedings IEEE . DOI: 10.1109/ISGTEurope.2017.8260283

# SDL::Establish Design Requirements

- Not only guide the implementation of design features, but also be resistant to known threats
- Saltzer and Schroeder security principles:
  - Economy of mechanism (KISS)
  - Fail-safe defaults (failure = lack of access)
  - Complete mediation (apply constant authorizations)
  - Open design (use of keys and passwords)
  - Separation of privilege (multiple keys, if possible)
  - Least privilege (only the needed set of privileges)
  - Least common mechanism (minimize common mechanisms)
  - Psychological acceptability (user interface shall help)

# SDL::Establish Design Requirements

- Complementary to Saltzer and Schroeder security principles:
  - Defense in depth (redundancy in security)
  - Design for updating (security patches shall be easy)
- Selection of security features, such as cryptography, authentication and logging to reduce the risks identified through threat modelling;
- Reduction of the attack surface (M. Howard, “Fending off future attacks by reducing attack surface,” MSDN Magazine, February 4, 2003. Accessed 12-10-2020 at [https://msdn.microsoft.com/en-us/library/ms972812.aspx](https://msdn.microsoft.com/en-us/library/ms972812.aspx))

# SDL::Establish Design Requirements

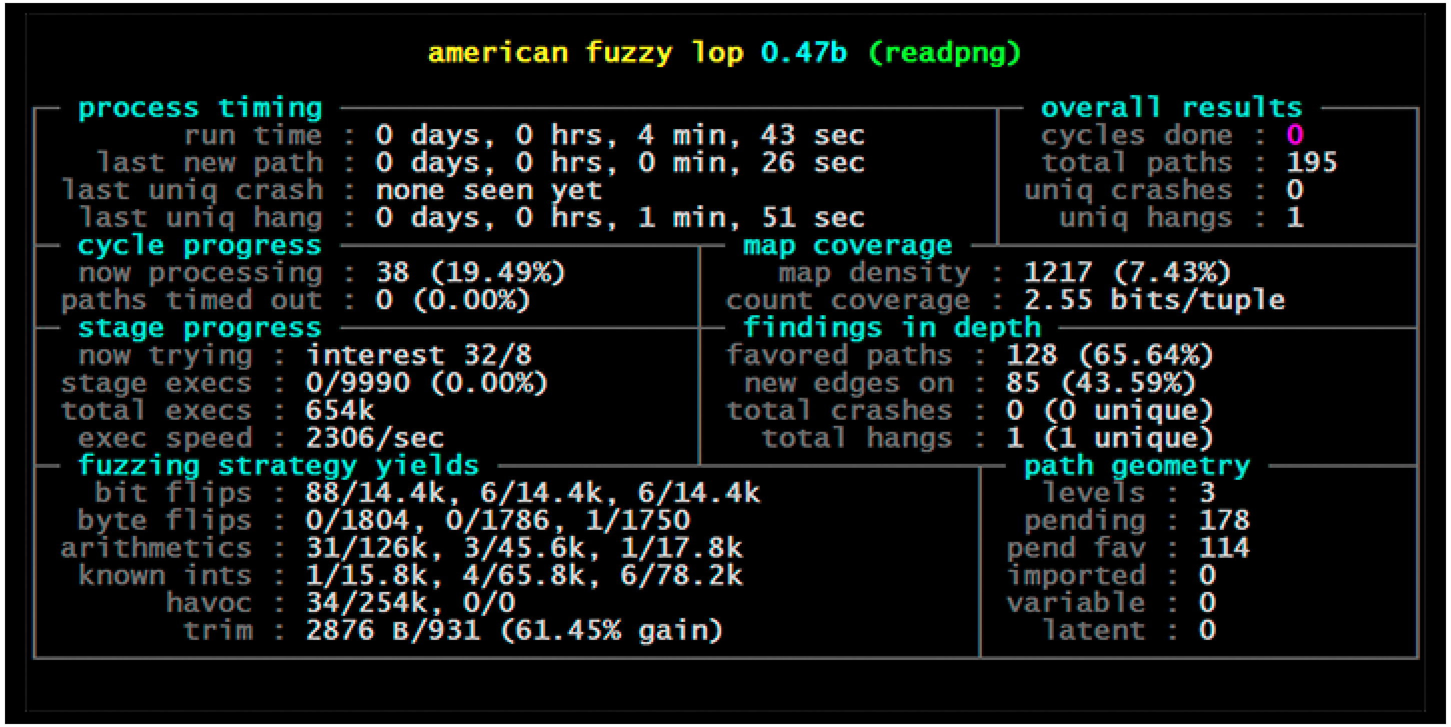
- IEEE Center for Secure Design top 10 security flaws:
  - Earn or give, but never assume, trust.
  - Use an authentication mechanism that cannot be bypassed or tampered with.
  - Authorize after you authenticate.
  - Strictly separate data and control instructions, and never process control instructions received from untrusted sources.
  - Define an approach that ensures all data are explicitly validated.
  - Use cryptography correctly.
  - Identify sensitive data and how they should be handled.
  - Always consider the users.
  - Understand how integrating external components changes your attack surface.
  - Be flexible when considering future changes to objects and actors.

# SDL::Perform Dynamic Analysis Security Testing (DAST)

- Run-time verification of compiled or packaged software to check functionality that is only apparent when all components are integrated and running.
- Use of a suite of pre-built attacks and malformed strings that can detect memory corruption, user privilege issues, injection attacks, and other critical security problems.
- May employ fuzzing, an automated technique of inputting known invalid and unexpected test cases at an application, often in large volume.
- Similar to SAST, can be run by the developer and/or integrated into the build and deployment pipeline as a check-in gate.
- DAST can be considered to be automated penetration testing.
- See also Section 3.2 (Dynamic Detection) in the Software Security knowledge area in the Cyber Security Body of Knowledge (see refs).

# SDL::Perform Dynamic Analysis Security Testing (DAST)

- Example of commonly used tool: https://lcamtuf.coredump.cx/afl/



```
                    american fuzzy lop 0.47b (readpng)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 4 min, 43 sec       │  cycles done : 0      │
│   last new path : 0 days, 0 hrs, 0 min, 26 sec       │  total paths : 195    │
│ last uniq crash : none seen yet                      │ uniq crashes : 0      │
│  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec       │   uniq hangs : 1      │
├─ cycle progress ──────────────────┬─ map coverage ───┴───────────────────────┤
│  now processing : 38 (19.49%)     │    map density : 1217 (7.43%)           │
│ paths timed out : 0 (0.00%)       │ count coverage : 2.55 bits/tuple        │
├─ stage progress ──────────────────┼─ findings in depth ─────────────────────┤
│  now trying : interest 32/8       │ favored paths : 128 (65.64%)            │
│ stage execs : 0/9990 (0.00%)      │  new edges on : 85 (43.59%)             │
│ total execs : 654k                │ total crashes : 0 (0 unique)            │
│  exec speed : 2306/sec            │   total hangs : 1 (1 unique)            │
├─ fuzzing strategy yields ─────────┴─────────────┬─ path geometry ───────────┤
│   bit flips : 88/14.4k, 6/14.4k, 6/14.4k        │    levels : 3             │
│  byte flips : 0/1804, 0/1786, 1/1750            │   pending : 178           │
│ arithmetics : 31/126k, 3/45.6k, 1/17.8k         │  pend fav : 114           │
│  known ints : 1/15.8k, 4/65.8k, 6/78.2k         │  imported : 0             │
│       havoc : 34/254k, 0/0                      │  variable : 0             │
│        trim : 2876 B/931 (61.45% gain)          │    latent : 0             │
└─────────────────────────────────────────────────┴───────────────────────────┘
```

# SDL::Perform Penetration Testing

- Penetration testing is black box testing of a running system to simulate the **actions of an attacker**.
- To be performed by **skilled security professionals**, internal or external to the organisation, opportunistically simulating the actions of a hacker.
- The objective is to uncover any form of vulnerability - from small implementation bugs to major design flaws resulting from coding errors, system configuration faults, design flaws or other operational deployment weaknesses.
- Tests should attempt both unauthorised misuse of and access to target assets and violations of the assumptions.
- A resource for structuring penetration tests is the OWASP Top 10 Most Critical Web Application Security Risks.
- Penetration testers can be referred to as **white hat hackers** or ethical hackers. In the penetration and patch model, penetration testing was the only line of security analysis prior to deploying a system.

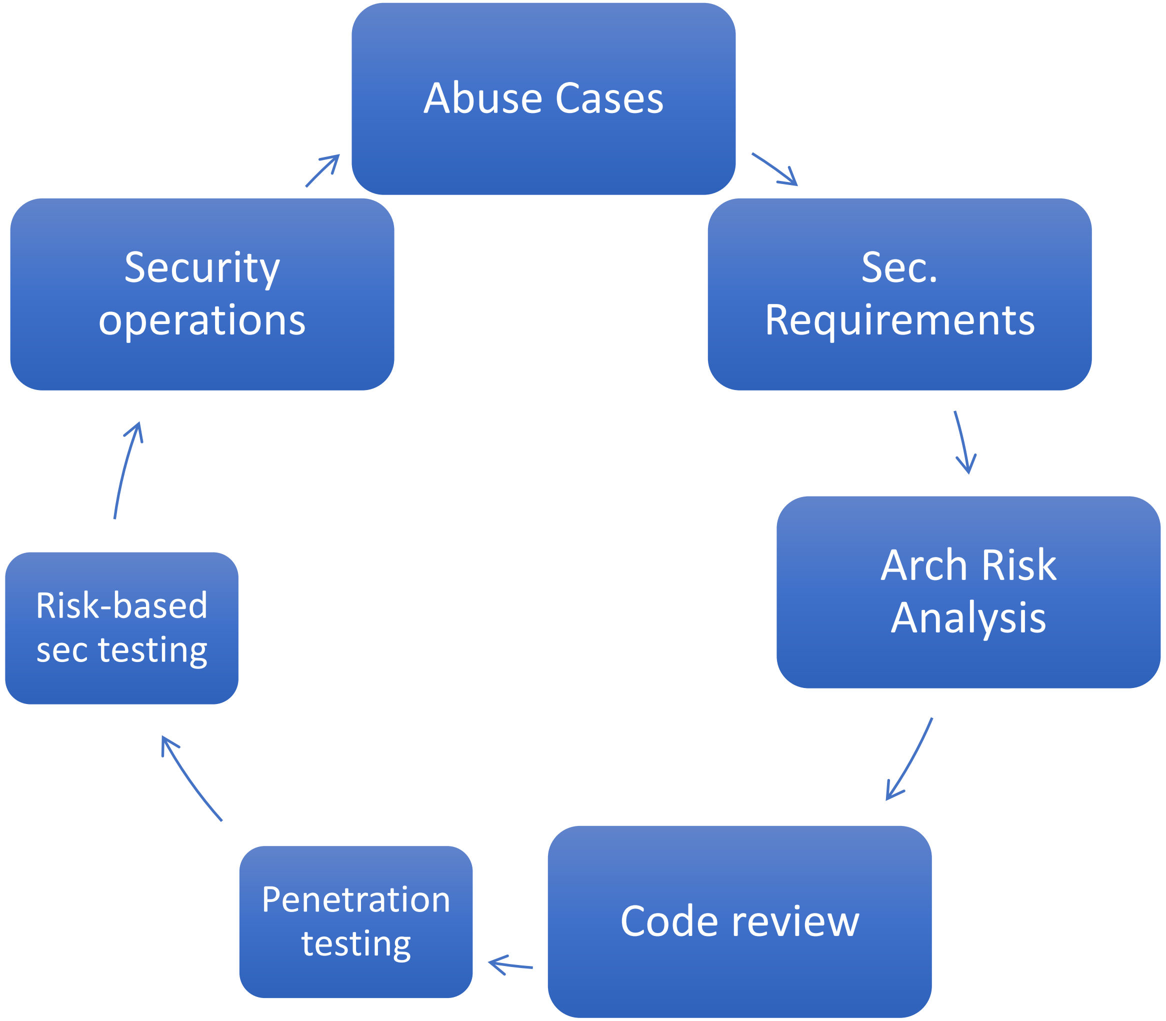
# SDL::Establish a Standard Incident Response Process

- Organisations must be prepared for inevitable attacks.
- Preparation of an Incident Response Plan (IRP).
- The plan shall include who to contact in case of a security emergency, establish the protocol for efficient vulnerability mitigation, for customer response and communication, and for the rapid deployment of a fix.
- It shall also include plans for code inherited from other groups within the organisation and for third-party code.
- The plan shall be tested before it is needed!
- Lessons learned (responses to actual attacks) → factored back into the SDL.

# Software Security Touchpoints

1. Code Review (Tools)
2. Architectural Risk Analysis
3. Penetration Testing
4. Risk-based Security Testing
5. Abuse Cases (thinking like an attacker)
6. Security Requirements
7. Security Operations (not only at software level)

# Touchpoints::Overview



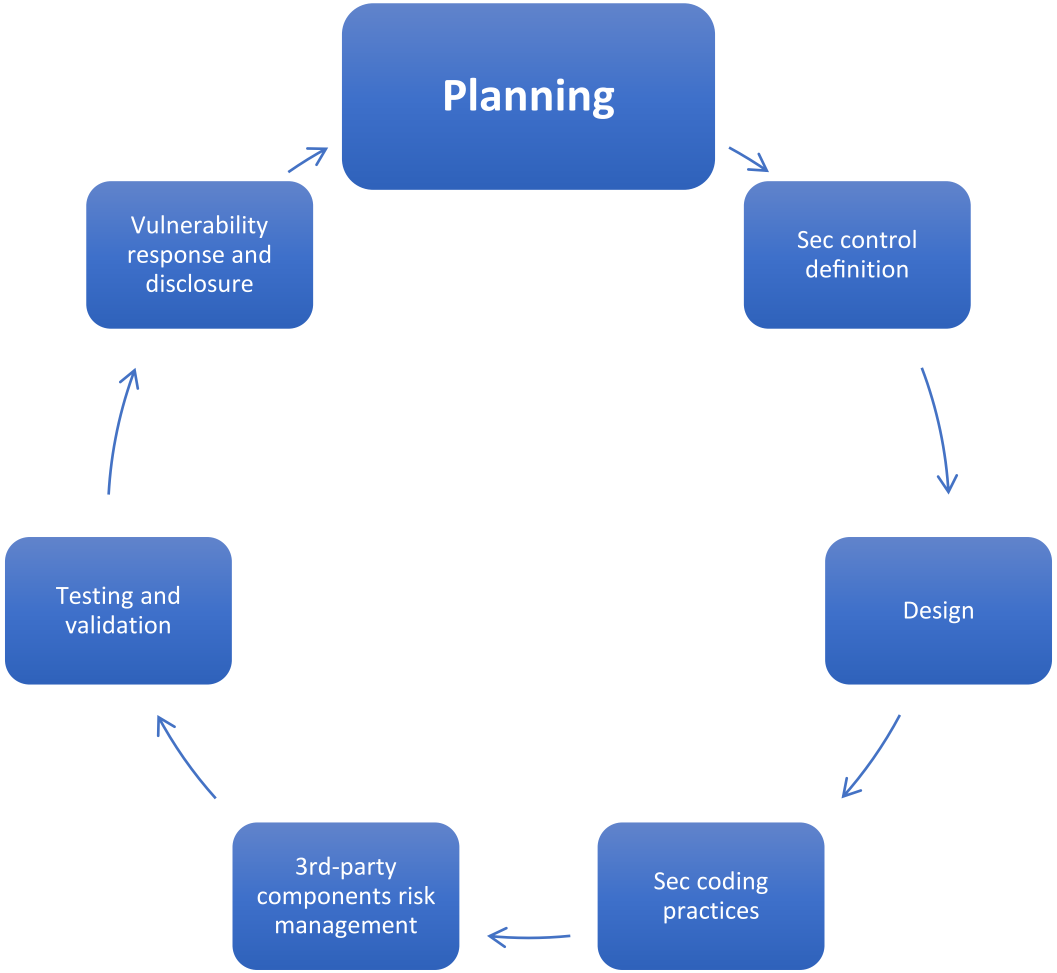
# Touchpoints::Architectural Risk Analysis

- Similar to Threat Modelling
- Designers and architects provide a high level view of the target system and documentation for assumptions, and identify possible attacks.
- McGraw proposes 3 main steps for risk analysis:
  - Attack resistance analysis (explore know threats)
  - Ambiguity analysis (discover new risks)
  - Weakness analysis (explore 3rd party assumptions)

# Software Assurance Forum for Excellence in Code (SAFECode)

1. Application Security Control Definition (security requirements)
2. Design
3. Secure Coding Practices (code standards, safe languages)
4. Manage Security Risk Inherent in the Use of 3rd party Components
5. Testing and Validation
6. Manage Security Findings (from previous steps)
7. Vulnerability Response and Disclosure (no perfectly secure product)
8. Planning the Implementation and Deployment of Secure Development (plan at organization level)

# SAFECode::Overview


Planning

Sec control definition

Design

Sec coding practices

3rd-party components risk management

Testing and validation

Vulnerability response and disclosure

# Secure SW Lifecycle Processes Summary

| Phase | Microsoft SDL | McGraw Touchpoints | SAFECode |
|---|---|---|---|
| Education and awareness | Provide training | | Planning the implementation and deployment of secure development |
| Project inception | Define metrics and compliance reporting<br>Define and use cryptography standards<br>Use approved tools | | Planning the implementation and deployment of secure development |
| Analysis and requirements | Define security requirements<br>Perform threat modelling | Abuse cases<br>Security requirements | Application security control definition |
| Architectural and detailed design | Establish design requirements | Architectural risk analysis | Design |
| Implementation and testing | Perform static analysis security testing (SAST)<br>Perform dynamic analysis security testing (DAST)<br>Perform penetration testing<br>Define and use cryptography standards<br>Manage the risk of using third-party components | Code review (tools)<br>Penetration testing<br>Risk-based security testing | Secure coding practices<br>Manage security risk inherent in the use of third-party components<br>Testing and validation |
| Release, deployment, and support | Establish a standard incident response process | Security operations | Vulnerability response and disclosure |

# Adaptations of the Secure Software Lifecycle

- CyBok (see references) contains hints for:
  - Agile Software Development and DevOps
  - Mobile
  - Cloud Computing
  - Internet of Things (IoT)
  - Road Vehicles
  - ECommerce/Payment Card Industry

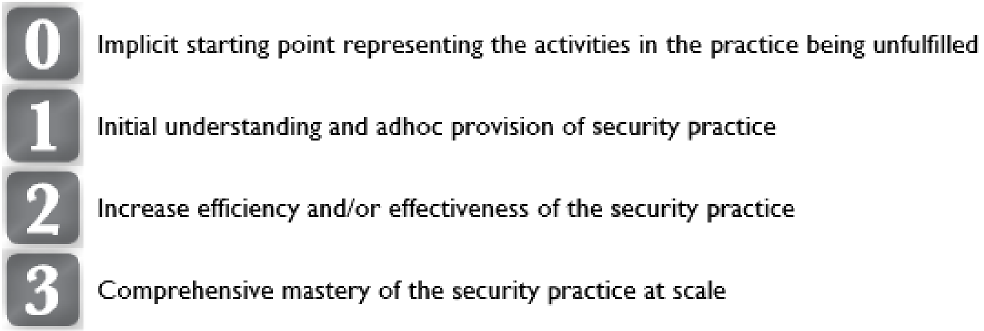
  - In summary it can be used almost in any type of project.

# Assessing the Secure Software Lifecycle

- There are different assessment approaches to evaluate the maturity of secure development lifecycle:
  - Software Assurance Maturity Model (SAMM)
  - Building Security In Maturity Model (BSIMM)
  - Common Criteria (CC)

# SAMM

- Assessment of a development process
  - 1. Define and measure security-related activities within an organisation.
  - 2. Evaluate their existing software security practices.
  - 3. Build a balanced software security program in well-defined iterations.
  - 4. Demonstrate improvements in a security assurance program.
- Uses 12 security practices grouped into one of 4 business functions:
  - Governance
  - Construction
  - Verification
  - Deployment

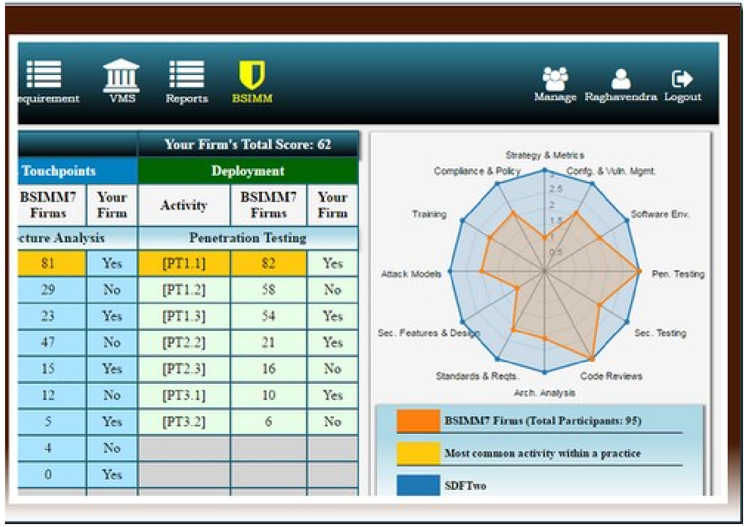


| | |
|---|---|
| 0 | Implicit starting point representing the activities in the practice being unfulfilled |
| 1 | Initial understanding and adhoc provision of security practice |
| 2 | Increase efficiency and/or effectiveness of the security practice |
| 3 | Comprehensive mastery of the security practice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

- Provides an organisation maturity level (0 to 3).
- Ref.: https://owasp.org/www-pdf-archive/SAMM_Core_V1-5_FINAL.pdf

# BSIMM

- Assessment of a development process based on SAMM
- Uses 12 security practices grouped into one of 4 business functions:
  - Governance
  - Intelligence
  - Secure software development lifecycle touchpoints
  - Deployment
- Provides comparison to other BSIMM assessed companies
- Ref.: [https://www.bsimm.com/](https://www.bsimm.com/)



Your Firm's Total Score: 62

| Touchpoints | | Deployment | | |
|---|---|---|---|---|
| BSIMM7 Firms | Your Firm | Activity | BSIMM7 Firms | Your Firm |
| cture Analysis | | Penetration Testing | | |
| 81 | Yes | [PT1.1] | 82 | Yes |
| 29 | No | [PT1.2] | 58 | No |
| 23 | Yes | [PT1.3] | 54 | Yes |
| 47 | No | [PT2.2] | 21 | Yes |
| 15 | Yes | [PT2.3] | 16 | No |
| 12 | No | [PT3.1] | 10 | Yes |
| 5 | Yes | [PT3.2] | 6 | No |
| 4 | No | | | |
| 0 | Yes | | | |

# CC

- Provides means for international recognition of a secure information technology
- Authorised Certification/Validation Body
- Reuse of Certified/Validates products with no further evaluation
- Based on Evaluation Assurance Levels (EAL):
  - 1 Functionally tested
  - 2 Structurally tested
  - 3 Methodically tested and checked
  - 4 Methodically designed, tested and reviewed
  - 5 Semi-formally designed and tested
  - 6 Semi-formally verified design and tested
  - 7 Formally verified design and tested

- Ref.: https://www.commoncriteriaportal.org/

# Recommendations

- **Think of security** early and often.
- Adopt a **software development model** to help define your organization's development activities and flow.
- **Define activities** for each phase in your model.
- Ensure all developers are **trained** to develop secure applications.
- **Validate** your software product at the end of every phase.
- Do not begin a software development project by writing code—**plan, specify and design first**.
- Keep the three SDL core concepts in focus—**education, continuous improvement, and accountability**.
- Develop **tests** to ensure each component of your application meets **security requirements**.

# References

- Trustworthy Software Foundation (https://tsfdn.org/resource-library/)
- US National Institute of Standards and Technology (NIST) NICE Cyber security Workforce Framework (https://www.nist.gov/itl/applied-cybersecurity/nice/resources/nice-cybersecurity-workforce-framework)
- Software Engineering Institute (SEI) (https://www.sei.cmu.edu/education-outreach/curricula/software-assurance/index.cfm)
- SAFECode free software security training courses (https://safecode.org/training/)

# References

- https://securityintelligence.com/series/ponemon-institute-cost-of-a-data-breach-2018/
- IEEE Center for Secure Design, “Avoiding the top 10 software security design flaws.” Accessed on 12-10-2020 at https://cybersecurity.ieee.org/blog/2015/11/13/avoiding-the-top-10-security-flaws/
- CyBOK Secure Software Lifecycle Knowledge Area Issue 1.0 © Crown Copyright, The National Cyber Security Centre 2019, licensed under the Open Government Licence http://www.nationalarchives.gov.uk/doc/open-government-licence/.

# The End

- Next up: Software Quality Attributes