The Microsoft SDL for Agile Development (Lecture 2 part 2)

Prof. Mathy Vanhoef

DistriNet – KU Leuven – Belgium

KU LEUVEN DistriNet

Agenda

- Types of Security Development Lifecycles (SDLs)
  - Including company-specific ones
- Microsoft SDL for agile development

Types of SDL models

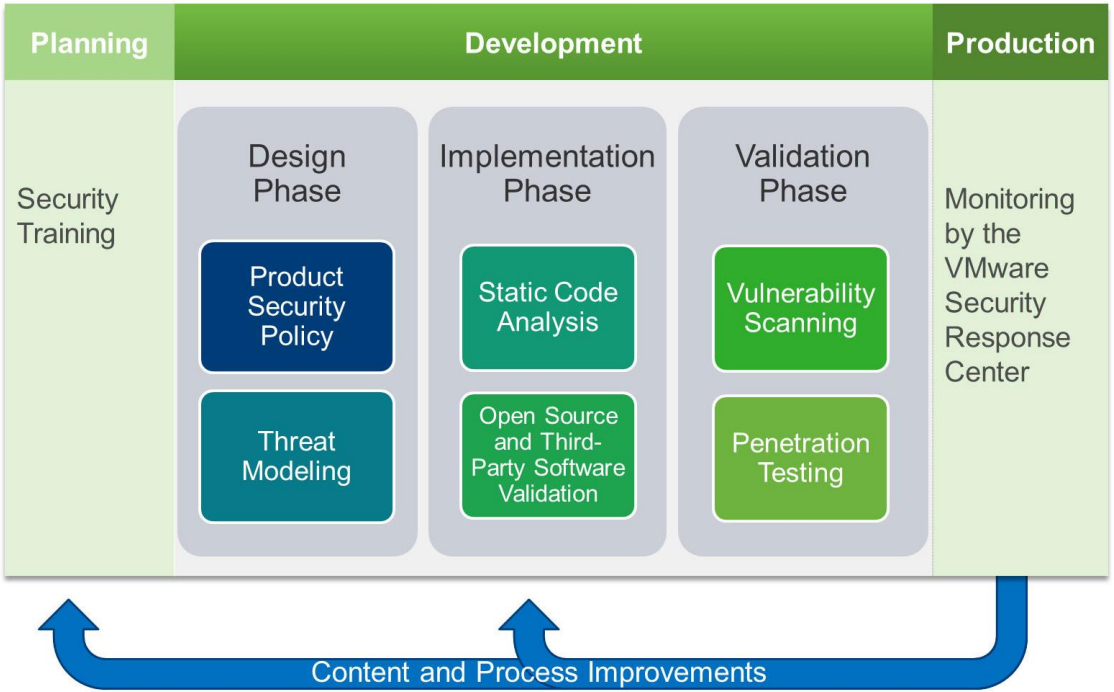
- **Process model:** adds security to standard development models
  - Microsoft SDL (old), NIST SP800-64, OWASP CLASP, SAFECode, TouchPoints
  - Usually based on waterfall/spiral model. Fewer models exist for Agile.*
- **General principles & best practices** collected in a document/model
  - Microsoft SDL (new), Generally Accepted System Security Principles (GASSP)
  - BSA Framework, PCI Framework, NIST SSDF
- **Maturity models:** know where you are & how to improve
  - OWASP Software Assurance Maturity Models (OWASP SAMM)
  - Building Security In Maturity Model (BSIMM)
  - System Security Engineering Capability Maturity Model (SSE-CMM)

* Source: “SDLC: Introduction and process models” by Bart De Win at SecAppDev, 2017.

Company-specific process SDLs

Some companies also (briefly) describe their own SDL variants

- **VMware**: regularly updates their SDL methodology based on evolving security.
- Not much details. “Follows standard security practices”

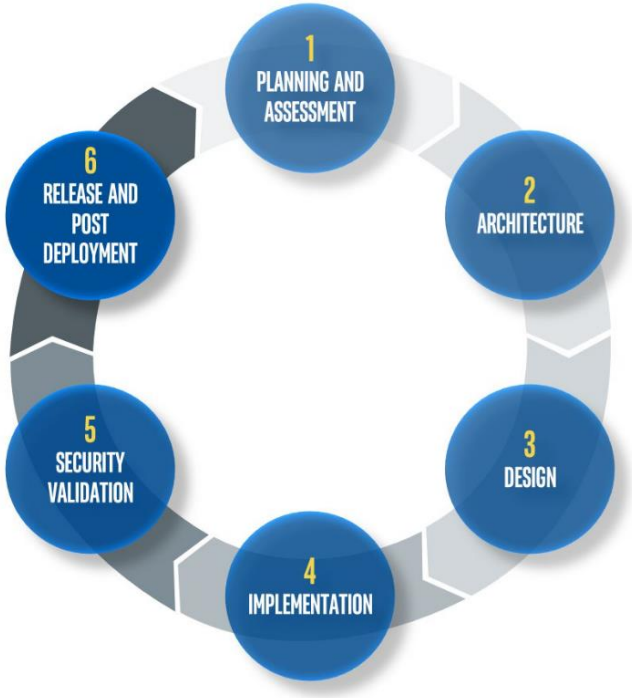


Source: https://www.vmware.com/nl/security/sdl.html

Company-specific process SDLs

Some companies also (briefly) describe their own SDL variants

- **Intel's SDL process**: applied across software, firmware, and hardware.
- Tackles challenges of hardware development, where hardware has longer development and support lifetimes than software.
- Mainly gives high-level info.



Source: https://newsroom.intel.com/wp-content/uploads/sites/11/2020/10/sdl-2020-whitepaper.pdf

And there are many more

Others that can be found online:

- Unity's S-SDLC: company-specific but quite detailed
- IEC 62443-4-1: for industrial automation & control systems
- ISO/IEC 27034: designed to help organizations build security throughout the life cycle of applications
- Cisco Secure Development Lifecycle
- Juniper Networks Secure Development Lifecycle
- And so on…

There is no “best” SDL model

What’s best depends on the lifecycle you already have, the type of product being made, regulatory requirements, etc.

Most SDLs contain similar core best practices and ideas

- The order and grouping of best practices may differ
- The level of detail and explanations given will also vary
- Each SDL typically has a different focus though

We will discuss some of the most well-known SDL models

Microsoft SDL for Agile

Microsoft SDL for Agile

Variant for Agile development:

- Security and privacy requirements are added to the product and sprint backlogs as tasks

A distinction is made between three types of requirements:

- Every-sprint requirements
- Bucket requirements
- One-time requirements

Every-sprint requirements

We can’t check/follow all requirements in every sprint

- But some requirements are so essential that no software should be released without these requirements being met.
- If these requirements are not met, the sprint is incomplete. E.g.:
  - Running an analysis tools daily or per build
  - Threat modelling of all new features
  - Ensuring that each project member has completed at least one security training in the past year
  - Use filtering and escaping libraries around all web output
  - Use only strong crypto in new code

Bucket requirements

- Requirements that must be performed on a regular basis
- Divided into three separate buckets of related tasks
- **Each sprint**, complete **one requirement from each bucket**

Verification tasks	Design Review	Planning
Network fuzzing	Privacy review	Create privacy documentation
Attack surface analysis	Review crypto design	Update security response contacts
BinScope analysis	Assembly naming & APTCA	Update network down plan
File fuzz testing	Use Account Control (UAC)	Define/update security bug bar

Bucket requirements

- Requirements that must be performed on a regular basis
- Divided into three separate buckets of related tasks
- **Each sprint**, complete **one requirement from each bucket**

Practical constraints:

- Methodology doesn’t mandate any type of round-robin or other task prioritization for these requirements.
- No bucket requirement can go more than six months without being completed.

One-time requirements

- Some tasks won’t need to be repeated once complete:
  - Create a baseline threat model
  - Establish a security response plan
  - Avoid writeable PE segments (more general: active compiler flags to enable low-level vulnerability mitigations)
  - Choosing a security advisor
  - Update your project to use the latest compiler version
- Tasks are typically short, but some can be longer
- Each one-time task must be completed in 1 month to 1 year

Constraints when using Agile

Summary: can’t follow all security requirements in one sprint

- With the bucket categorization, you can skip some security requirements from some releases.
- This is necessary to balance security, feature development, and speed of release when using short release cycles.

Let’s now discuss more how to execute some of the tasks

Applying task: security education (every-sprint)

- Every project member must complete one security training course every year.
- If more than 20% of members are out of compliance, this requirement is failed.
  - As a result, the sprint is also failed and the product is not allowed to be released.
- Training can be as simple as reading appropriate chapters in a book or watching an online training class

Applying task: threat modeling (every-sprint)

- Threat model process can be time-boxed and limited to the parts of the product that currently exist or are in development
- During each sprint, the threat model should be updated to represent new functionality added during that sprint
- This is an essential task and done in every sprint
  - It’s hard to automate this task. But only new features/changes need to be threat modeled.

Applying task: final security review (every-sprint)

Final security review is required at the **end of every sprint**

- In SDL-Agile, this review is limited in scope. Check that:
  - **All every-sprint** requirements have been completed
  - No security bugs are open that fall above the **security bug bar**
  - At least one requirement from **each bucket** has been completed
  - No bucket requirement has gone uncompleted for more than 6 months
  - No one-time requirements have exceeded their grace period

= Check whether processes are followed (no code audits)

Optional extra: using a Spike

A **spike** = a sprint totally focused on security:

- Time-boxed “side project” to find security bugs
- Typically done to audit **older or legacy code** since those tend to have more security bugs
- This is similar to a code push: bring risky code up to date in a short amount of time
- The spike doesn’t propose fixing the bugs yet but rather analyzing them to determine how bad they are

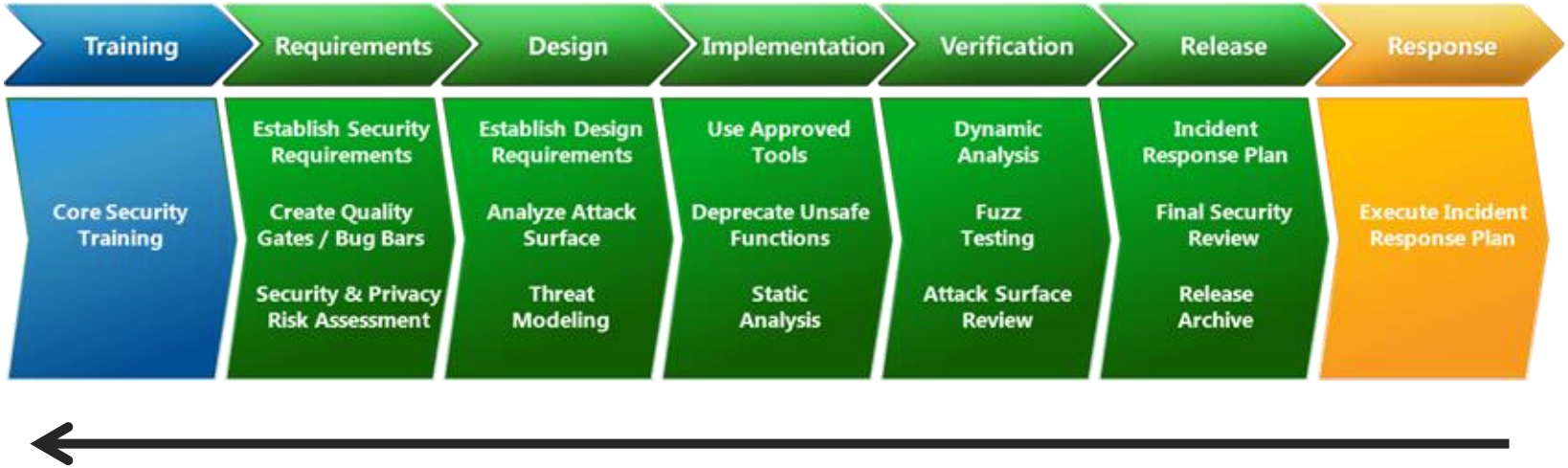
Example of SDL-Agile

See the “SDL Process Guidance Version 5.2” for an example on how sprints can be incorporate security like we discussed.

- Hypothetical scenario where a database-driven web project is developed by a team with four-week sprints.

Summary

- Security must be considered in every phase of the lifecycle
- Must start with security early, called the **shift left trend**:



Shift-left: assuring security early in the software lifecycle

References

Required reading:

- Simplified Implementation of the Microsoft SDL, 2012.

Optional reading:

- Microsoft Security Development Lifecycle (SDL) Process Guidance - Version 5.2 Detailed explanation of the SDL that was covered in this lecture (including more examples).
- The Security Development Lifecycle book by Michael Howard and Steve Lipner, 2006.

Optional extra information

- “SDL That Won’t Break the Bank” by Steve Lipner, Black Hat
  - Slides and presentation are online, 2019
- “Secure Development Lifecycles (SDLC): Introduction and Process Models” by Bart De Win, SecAppDev
  - Slides (2019) and older presentation (2017)
- “Security in a fast moving Agile/DevOps Environment” by Bart De Win, SecAppDev.
  - Slides only (2019)