UNIVERSIDAD TECNOLÓGICA DE
SAN LUIS RIO COLORADO

Secure Software Development Lifecycle

MTRO. Aurelio Flores

ALUMNO: VICTOR MANUEL GALVAN COVARRUBIAS

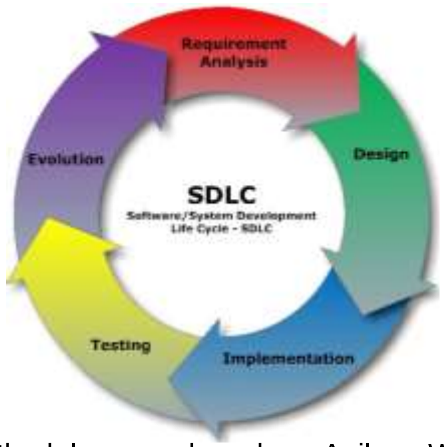
ING. EN DESARROLLO Y GESTIÓN DE SOFTWARE

San Luis Rio Colorado, Sonora Mayo, 2022

Secure Software Development Lifecycle is the process of including security artifacts in the Software Development Lifecycle. Consists of a detailed plan that defines the process organizations use to build an application from inception.

Development teams use different models. However, all models usually follow these phases:

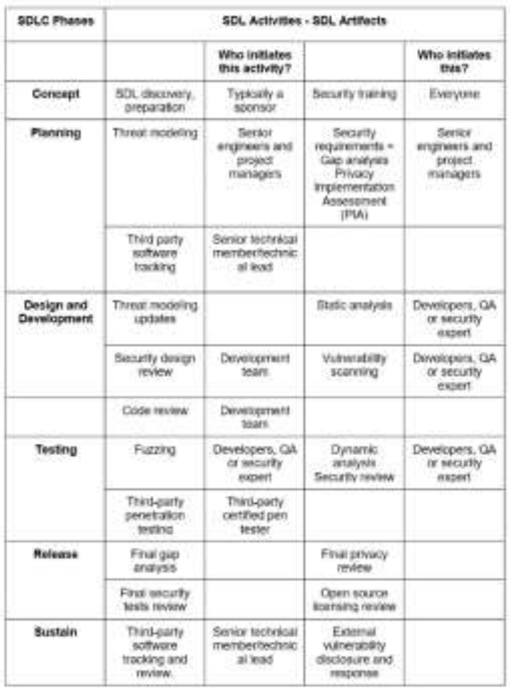
- Plan and requirements
- Design
- Test plan
- Coding
- Testing
- Release
- Maintenance



There are seven phases in most SDLCs,
although they may vary according to the methodology used, such as Agile or Waterfall:

- Concept
- Planning
- Design and Development
- Testing
- Release
- Sustain
- Disposal

Each security activity should correspond with a phase in the SDLC



SDLC Phases	SDL Activities - SDL Artifacts			
		Who initiates this activity?		Who initiates this?
Concept	SDL discovery, preparation	Typically a sponsor	Security training	Everyone
Planning	Threat modeling	Senior engineers and project managers	Security requirements = Gap analysis Privacy Implementation Assessment (PIA)	Senior engineers and project managers
	Third party software tracking	Senior technical member/technical lead		
Design and Development	Threat modeling updates		Static analysis	Developers, QA or security expert
	Security design review	Development team	Vulnerability scanning	Developers, QA or security expert
	Code review	Development team		
Testing	Fuzzing	Developers, QA or security expert	Dynamic analysis Security review	Developers, QA or security expert
	Third-party penetration testing	Third-party certified pen tester		
Release	Final gap analysis		Final privacy review	
	Final security tests review		Open source licensing review	
Sustain	Third-party software tracking and review.	Senior technical member/technical lead	External vulnerability disclosure and response	

Security testing checks how vulnerable is the new product to attacks. The activities include:

- **Static Analysis**: identifies the exact location of weaknesses by analyzing the software without executing it.
- **Dynamic Analysis**: identifies weaknesses by running the software, helping find infrastructure flaws and patch errors.
- **Vulnerability Scanning**: injects malicious inputs against running software to check how the program reacts.
- **Fuzzing**: involves giving invalid, random data to a program, to check for access protocols and file formats
- **Third-party penetration testing**: the tester simulates an attack to discover coding or system configuration flaws, and discover vulnerabilities a real attacker can exploit. It is required that the tester is an external party not connected to the team.

Phases of Secure Software Development Life Cycle

- **Requirements**

In this early phase, requirements for new features are collected from various stakeholders. It's important to identify any security considerations for functional requirements being gathered for the new release.

- **Design**

This phase translates in-scope requirements into a plan of what this should look like in the actual application. Here, functional requirements typically describe what should happen, while security requirements usually focus on what shouldn't.

- **Development**

There are usually established secure coding guidelines as well as code reviews that double-check that these guidelines have been followed correctly. These code reviews can be either manual or automated using technologies such as static application security testing.

- **Verification**

Applications go through a thorough testing cycle to ensure they meet the original design & requirements. This is also a great place to introduce automated security testing using a variety of technologies.

- **Maintenance and evolution**

Vulnerabilities that slipped through the cracks may be found in the application long after it's been released. These vulnerabilities may be in the code developers wrote, but are increasingly found in the underlying open-source components that comprise an application.