Mestrado em Engenharia Informática (MEI)
Mestrado Integrado em Engenharia Informática (MiEI)

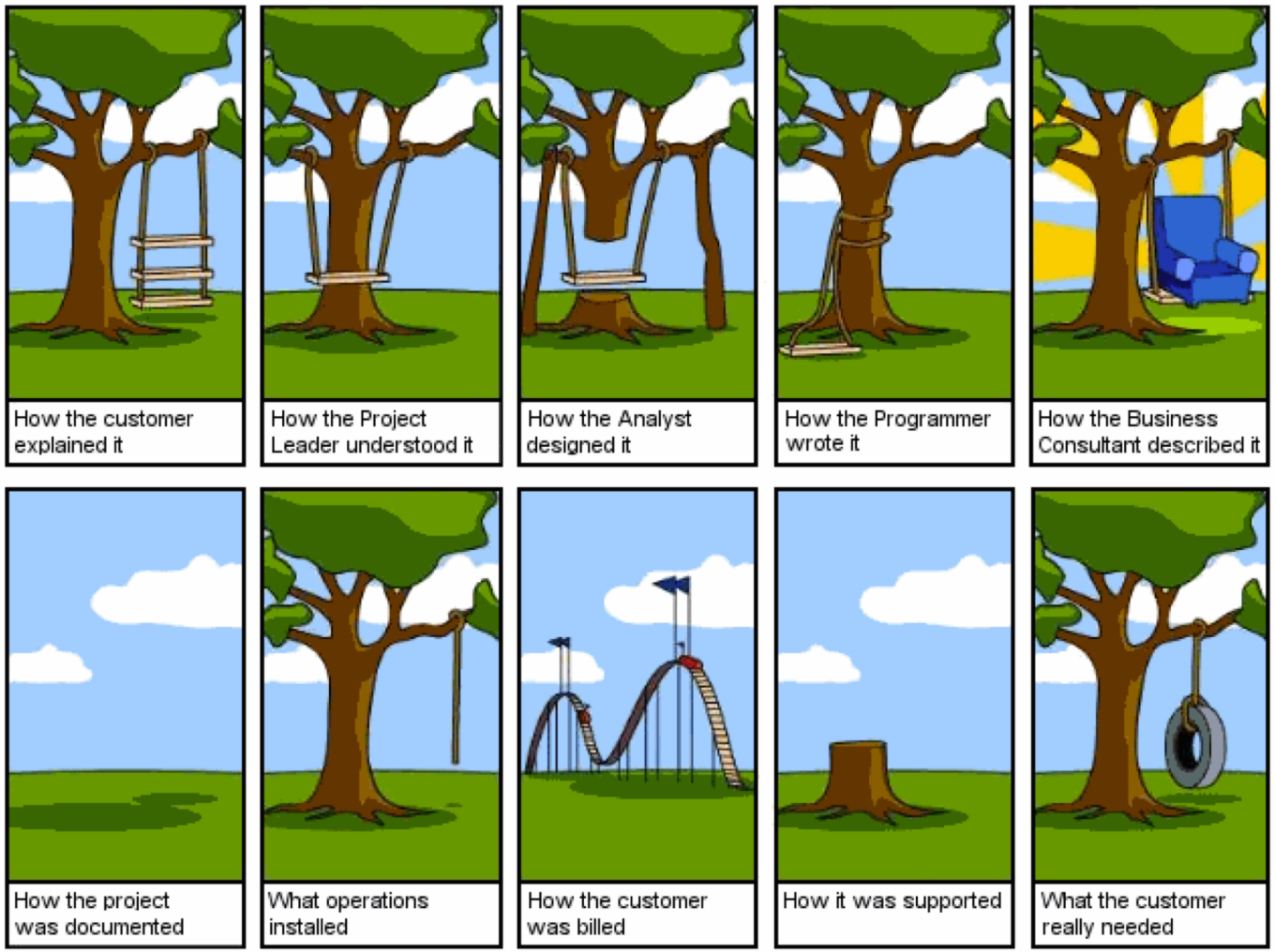
Perfil de Especialização **CSI** : Criptografia e Segurança da Informação

Engenharia de Segurança

jose.miranda@devisefutures.com

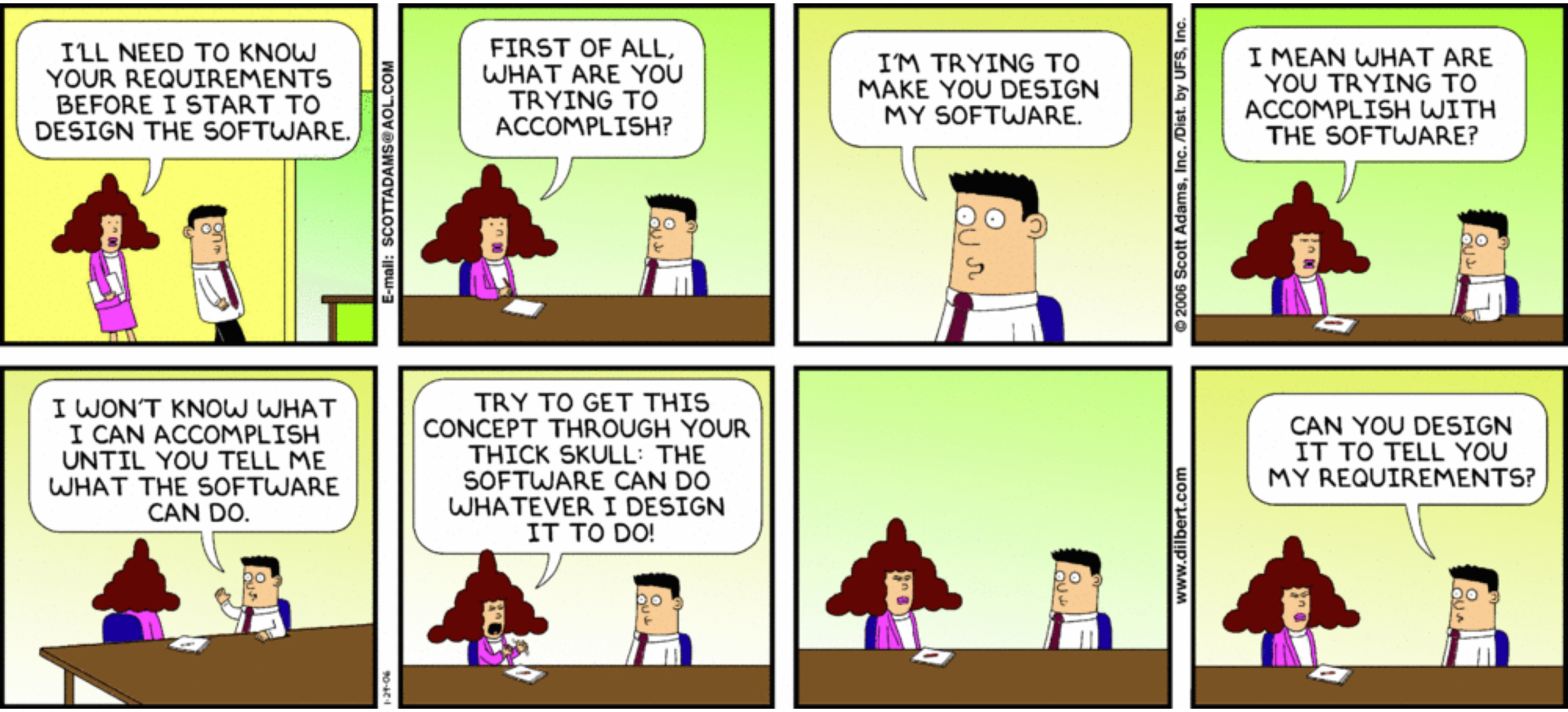
Topics

- *Secure Software Development Lifecycle* (S-SDLC)



Topics

- *Secure Software Development Lifecycle* (S-SDLC)



Topics

- *Secure Software Development Lifecycle* (S-SDLC)

Software Life Cycle Process

- What is *Software Life Cycle* Process?
  - Process used by the software industry to acquire, supply, develop, operate and maintain software;
  - It **aims** to produce high quality software that meets or exceeds customer's expectations and is finalized within time and cost estimate;
  - **ISO / IEC / IEEE 12207: 2017** (*Systems and software engineering -- Software life cycle processes*) defines software engineering processes, activities and tasks associated with the software life cycle from design to software withdrawal / discontinuation, adapted to each software project.
    - **Key processes**: Acquisition, Supply, Development, Operation, Maintenance.
    - **Support Processes**: Documentation, Configuration Management, Quality Assurance, Verification, Validation, Joint Review, Audit, Troubleshooting, Usability, Change Request Management, Product Evaluation.
    - **Organizational processes**: Management, Infrastructure, Improvement, Human Resources.
    - **Software reuse processes**: Asset management, Reuse program management, Maintenance engineering.
    - **Adaptation processes**, within the scope of Project, Organization, Culture, Life cycle model, methods and techniques, and languages.

Software Life Cycle Process

- Software life cycle (ISO/IEC/IEEE 12207:2017) **Key processes:**
  - **Acquisition** - all activities to obtain the product / service that satisfies the needs of the contracting company;
  - **Supply** - development of the project management plan, with the different steps / milestones, to be used in the development process;
  - **Development** - design, creation and testing of the product / software system, resulting in a software product / system suitable to be delivered to the client;
  - **Operation** - activities required to use the product / software system;
  - **Maintenance** - activities required to keep the product / system running.

Software Life Cycle Process

- Core processes of the software life cycle (ISO / IEC / IEEE 12207: 2017)
  - **Acquisition** - all activities to obtain the product / service that satisfies the needs of the contracting company:
    - **Start and scope**: Description of the need to acquire, develop or improve a software product / system; Definition and approval of the requirements of the product / software system; Evaluation of alternatives (purchase of off-the-shelf product, specific development or improvement of existing product); Development of acquisition plan; Definition of the acceptance criteria of the product / software system;
    - **Preparation of the request for proposal**: Contains requirements of the software product / system and technical restrictions (interoperability with other systems, environment in which it will be used, ...), delivery milestones and acceptance criteria;
    - **Preparation of the contract**: Development of supplier selection procedure; Selection of suppliers, based on the selection procedure; Initial version of the contract;
    - **Negotiation of changes**: Negotiation with selected suppliers;
    - **Contract update**: Contract is updated as a result of previous activity negotiations;
    - **Monitoring of the supplier**: Monitoring the activity of the suppliers according to the steps / milestones contracted (if necessary, working directly with suppliers to ensure timely deliveries);
    - **Acceptance and finalization**: Development of acceptance tests and procedures; Acceptance testing; Conduct product / system configuration management.

Software Life Cycle Process

- Core processes of the software life cycle (ISO / IEC / IEEE 12207: 2017)
  - **Development** - design, creation and testing of the product / software system, resulting in a software product / system suitable to be delivered to the cliente:

Requirements Analysis

- **Requirements definition** - Obtain and define the requirements and requests of the client through direct request or through the request for proposal. It is imperative to understand the customer's expectations and to ensure that both the customer and the supplier understand the requirements in the same way. It is important to manage all changes made to customer requirements in relation to the defined baseline ensuring that the outcome of technological changes and customer needs are identified and the impacts of introducing those changes are evaluated.
- **Specification of functional requirements** - Definition of the product / software system functional requirements (System functionalities and capabilities; Business, organizational and user requirements; Safety, ergonomics, interface, operation and maintenance requirements; project and certification / accreditation requirements);

Design

- **High level architecture** - Design of the basic organization of the product / system, i.e. identification of the different modules (hardware, software and manual operations) and how they communicate with each other (without great detail of the modules);
- **Design of modules** - The different modules are designed separately, with as much detail as possible;

Implementation

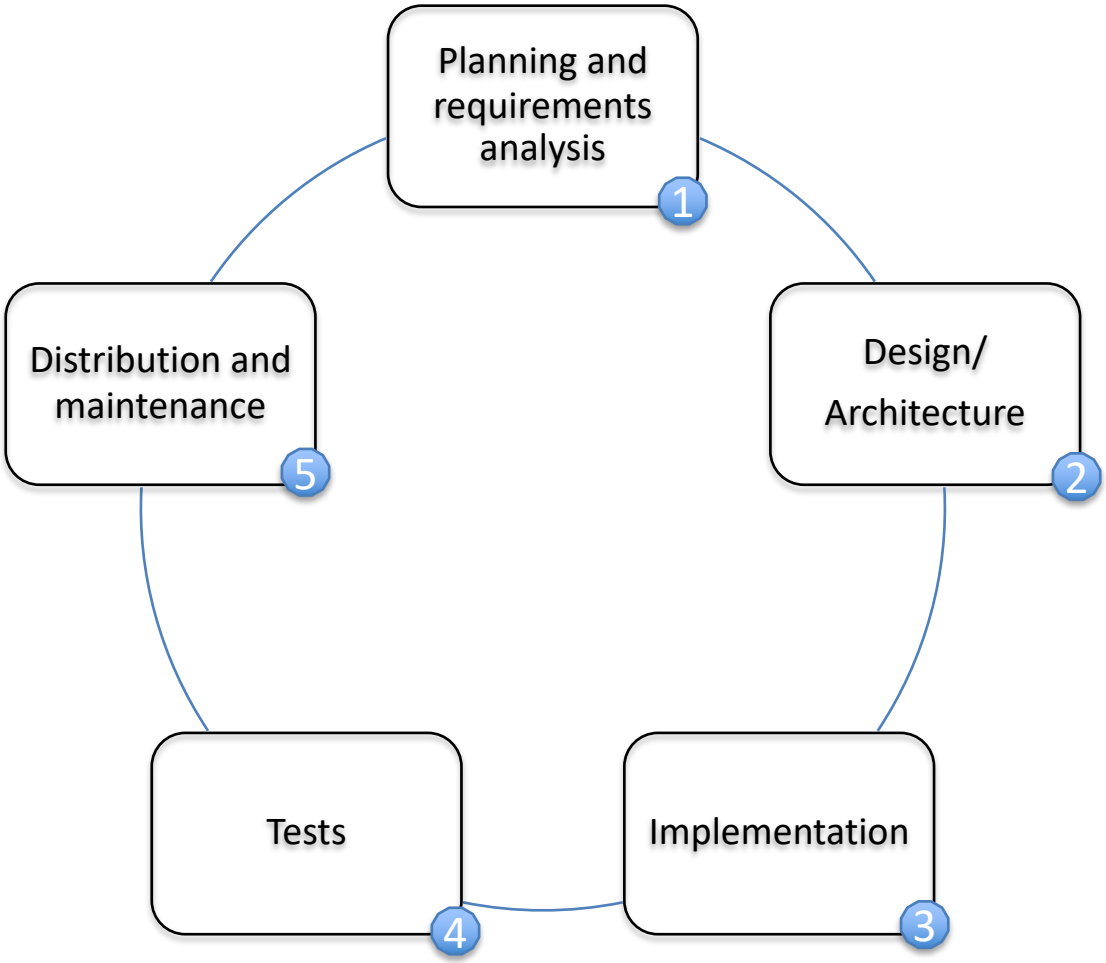
- **Coding / Implementation** - Creation of the code according to the high level architecture and design of the modules; Execution of change control; Documentation and resolution of nonconformities and problems encountered; Selection, adaptation and use of standards, methods, tools and programming languages.

Tests

- **Modules test** - Testing the correct operation of each of the different modules;
- **Integration test** - Test of the correct communication between modules;
- **System Test** - Test that all functional requirements are present in the product / software system. In that case, the product is finalized and can be delivered to the customer;
- **Installation of the product / software system** - Installation of the product / software system in the target environment; Configuration and initialization of the product / software system and other necessary systems (for example, database);
- **Support to the acceptance of the software product / system** - Support to the acceptance of the software by the customer, including customer testing.

Software Development Lifecycle (SDLC)

Common phases of *Software Development Lifecycle* (SDLC)

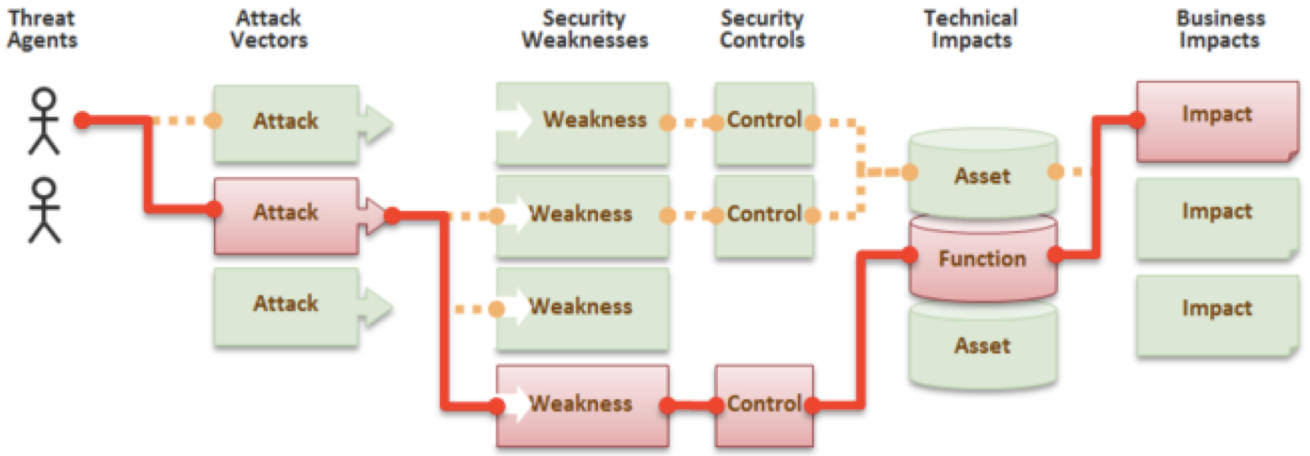


Software Development Lifecycle (SDLC)

- Software security flaws can be introduced at any phase of the development cycle:
  - At first, by failing to identify the security needs
  - In the creation of conceptual architectures that have logic errors
  - In the use of bad programming practices that introduce technical vulnerabilities
  - In improperly implementing the software
  - Inserting failures during maintenance or upgrade
- Software vulnerabilities may result from a much larger scope than the software itself. Among others:
  - Third-party APIs used by software
  - The application servers (and others)
  - The operating system of the servers
  - The backend database
  - Other applications in a shared environment
  - The user system
  - Other software that the user interacts with

Software Development Lifecycle (SDLC)

- Facts
  - A development team develops an application to perform specific tasks based on functional requirements and documented use cases.
  - An attacker is more interested in what the application can be made to do and assumes that "any action not expressly prohibited is permitted."
  - Attackers can use many "paths" to attack the business or organization.
  - "Paths" can be trivial to find and explore or can be extremely difficult. The damage done may be null or void the company business.

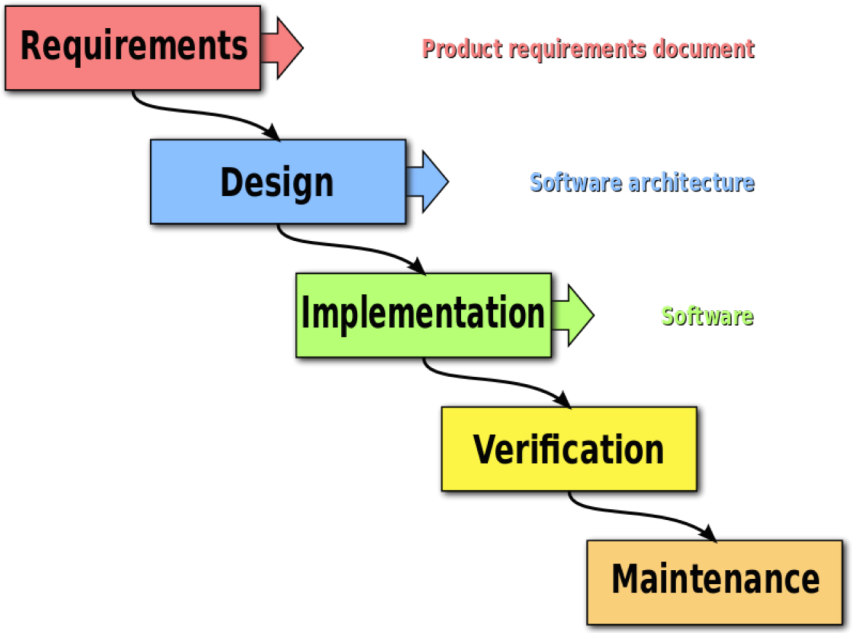


Secure Software Development Lifecycle (S-SDLC)

- Several ways to add security to the SDLC:
  - **Add security to conventional software development models;**
  - Use a secure software development model.
  - By adding security using a Maturity Model.

Secure Software Development Lifecycle (S-SDLC)

- One way to add security to the SDLC is to add security to conventional software development models.
  - Waterfall model



https://en.wikipedia.org/wiki/Waterfall_model

Secure Software Development Lifecycle (S-SDLC)

- One way to add security to the SDLC is to add security to conventional software development models.
  - **Waterfall model**
    - Security must be considered in the different phases of the model
    - **Requirements Phase**
      - Security requirements should be based on existing legislation (eg, Sarbanes-Oxley Act, Health Insurance Portability and Accountability Act (HIPAA), GPDR, ...) and international recommendations and standards (ISO 27000 family), as applicable, and should be translated into specific requirements for the software to be developed.
    - **Design Phase**
      - The security requirements defined in the previous phase are translated into mechanisms that implement them (authentication mechanisms, protection of confidentiality and data integrity, ...).
    - **Implementation Phase**
      - Secure programming, for which it is necessary to know the various types of coding vulnerabilities, and static code analysis tools must be used.
    - **Test Phase**
      - Carry out validations and security tests
    - **Maintenance Phase**
      - Security measures in the distribution of patches, product updates / upgrades, ...

Project Vulnerabilities

Erros in this phases

Implementation Vulnerabilities

Errors in this phase

Secure Software Development Lifecycle (S-SDLC)

- Several ways to add security to the SDLC:
  - Add security to conventional software development models;
  - **Use a secure software development model.**
  - By adding security using a Maturity Model.

Secure Software Development Lifecycle (S-SDLC)

- Secure Software Development Model
  - **Microsoft Security Development Lifecycle**
    - **(Pre)Training Phase**
      - Preparation of the members of the software development team to deal with the security aspects
    - **Requirements Phase**
      - Identify the people responsible for coordinating project security issues;
      - Identify a bug tracking tool, appropriate to vulnerability management;
      - Define minimum security requirements, similar to what was defined for the requirements phase of the cascade model
    - **Design Phase**
      - Designing the security mechanisms to implement, following recommendations and best practices, and avoiding project vulnerabilities;
      - Conduct security risk analysis to identify the most critical from the point of view of security and privacy.
    - **Implementation Phase**
      - Define and follow good coding practices;
      - Document how users should configure the software securely.
    - **Verification Phase**
      - Objective is to ensure that the software holds the desired security properties, through manual code review, test and static code analysis.
    - **Release Phase**
      - Plan actions to take when vulnerabilities are discovered;
      - Prior to publication, a final and independent security review should be carried out.
    - **(Post)Response Phase**
      - Recolha de informação sobre incidentes de segurança e, criação de relatórios sobre vulnerabilidades e *patches*.

Secure Software Development Lifecycle (S-SDLC)

In any of the S-SDLC it's fundamental in the **Requirements phase**:

- Identify the security requirements
  - Software requirements indicate what the software should do while the security requirements indicate the security expectations in the execution / use of the software;
  - Two types of security requirements:
    - Security related objectives (security policies):
      - Confidentiality (and Privacy and Anonymity)
        - » Policy Example: the Bank account is known only to its legitimate owner
      - Integrity
        - » Policy Example: Account operations must be authorized by the account owner
      - Availability
        - » Policy Example: The user can always access to obtain the balance of his account

Secure Software Development Lifecycle (S-SDLC)

- Identify the security requirements
  - Software requirements indicate what the software should do while the security requirements indicate the security expectations in the execution / use of the software;
  - Two types of security requirements:
    - Security related objectives (security policies):
    - Required mechanisms to enforce requirements:
      - Authentication
        - » What the user knows (e.g., password),
        - » What the user is (e.g., fingerprint),
        - » What the user has (e.g., citizen card)
      - Authorization
        - » According to policies that define actions that can be authorized (access control, ...)
      - Auditability
        - » Save enough information to determine the circumstances of an action (usually in logs, protected against forgery)

Secure Software Development Lifecycle (S-SDLC)

- <u>Identify the security requirements</u>
  - What security requirements have to be followed?
    - Based on legislation applicable to the business;
    - Based on standards;
    - Based on company values and / or policies;
    - Based on risk models;
    - Based on abuse cases.

Secure Software Development Lifecycle (S-SDLC)

In any of the S-SDLC it's fundamental in the **Requirements phase**:

- Identify Abuse Cases
  - Illustrate Security Requirements;
  - Describe what should **not** be done and / or allowed;
  - Based on attack patterns and likely scenarios, build abuse cases in which an attacker may violate security requirements.
    - For example, an attacker steals the password file and gets all user passwords
    - Another example, an attacker repeats a money transfer message

Secure Software Development Lifecycle (S-SDLC)

In any of the S-SDLC it's fundamental in the **Requirements phase**:

- Architecture Risk Analysis
  - Risk assessment process of a security breach, based on the probability and cost of various types of attacks, and doing its utmost to minimize this risk;
  - Based on risk / threat models:
    - **Attacker on the network** - anonymous user who can access the service through the network - can:
      - Measure the size of the requests and the time of the answers;
      - Perform parallel sessions;
      - Provide poorly formatted inputs and messages;
      - Remove or send extra network packets;
    - **Attacker on the same network** as the user of a particular service - for example, connected to an insecure Wi-Fi network, in a shopping center - which in addition can:
      - Read and measure messages from third parties;
      - Intercept, duplicate and modify messages;

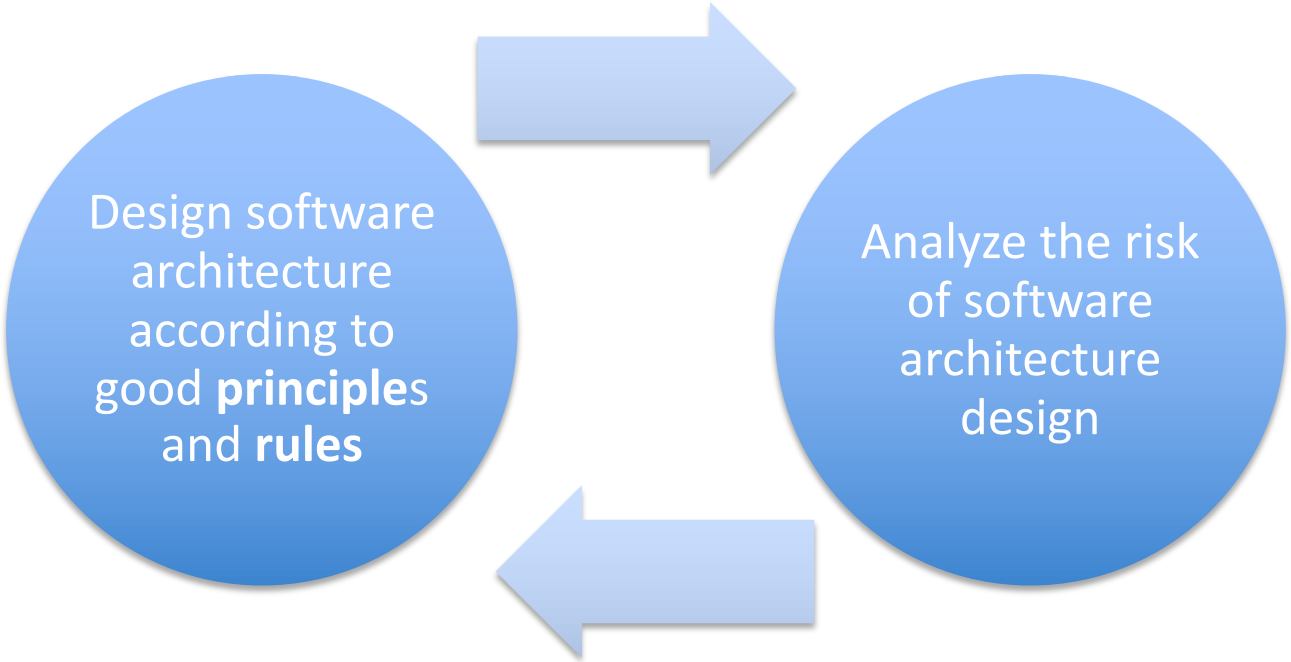
Secure Software Development Lifecycle (S-SDLC)

- Architecture Risk Analysis
  - Based on risk / threat models:
    - **Attacker on the same machine** as the user of a particular service - for example, malware installed on the laptop - which in addition can:
      - Read and write files (including cookies) and user memory;
      - Spy on what is written and other events;
      - Read and write on the user's screen.
  - Different risk model considered leads to different types of responses in software analysis and design;
  - The **essence of risk analysis** is to take into account the potential **costs** of developing the response to a given type of risk, the **costs** of a successful security breach, and weighing the **likelihood** of each.

Secure Software Development Lifecycle (S-SDLC)

In any of the S-SDLC it's fundamental in the **Requirements phase**:

- Security-oriented Design
  - Iterative process



Secure Software Development Lifecycle (S-SDLC)

- Security-oriented Design
  - Categories of Principles:
    - Prevention - completely eliminate software defects;
    - Damage reduction - reduce damage from exploitation of unknown defects;
    - Detection (and recovery) - identify and perceive an attack (and undo damage).
  - Principles:
    - Favor simplicity
      - Use fail-safe defaults
      - Do not expect customers to be experts
    - Trust reluctantly
      - Grant the minimum privilege ("need to know")
    - Defend in depth
      - Avoid dependence on only one defense mechanism (security by diversity);
      - No security by obscurity
      - Keeping up-to-date on the latest threats and researches
    - Monitoring and Traceability
      - Detect and diagnose security breaches and / or attacks

Secure Software Development Lifecycle (S-SDLC)

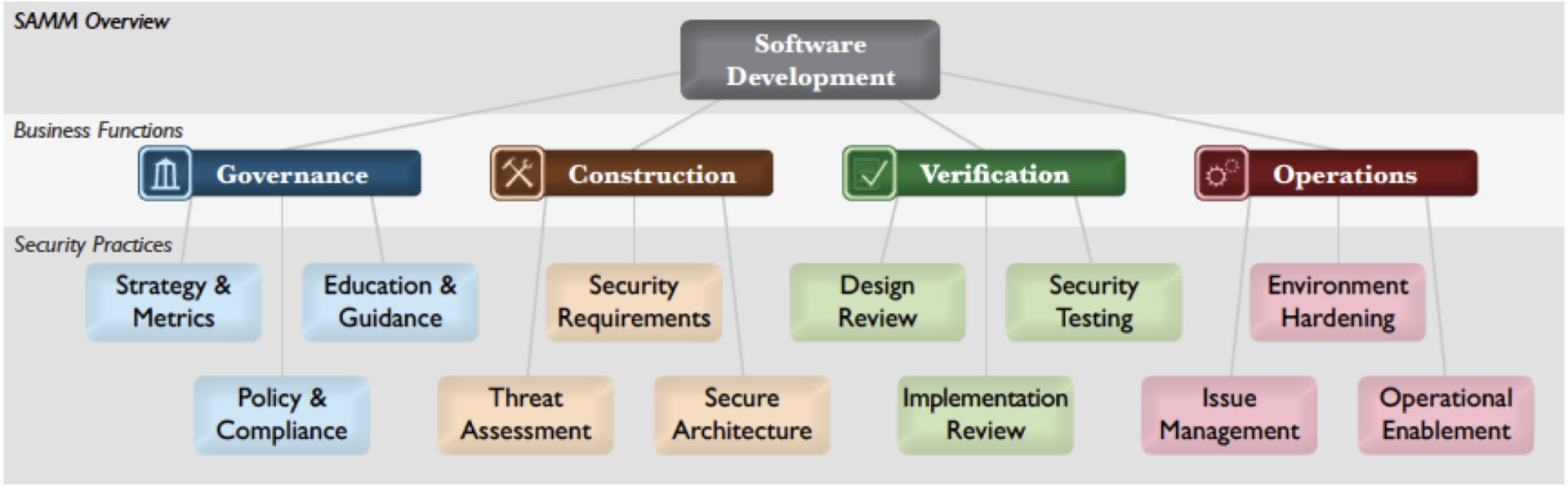
- Several ways to add security to the SDLC:
  - Add security to conventional software development models;
  - Use a secure software development model.
  - **By adding security using a Maturity Model.**

Secure Software Development Lifecycle (S-SDLC)

- Through the addition of security using a Maturity Model.
  - The behavior of organizations changes slowly, so changes must be iterative, according to long-term objectives (development of secure software);
  - There is no recipe that works for all organizations, so the organization must be able to choose the risks it wants to run;
  - Need to evaluate existing software security practices in the organization and create a software security program that allows to achieve the organization's goals in well-defined iterations;
  - The guide to security activities to consider in software development should provide sufficient detail for people outside the security area;
  - In general, the model to be used for the development of secure software must be simple, well-defined and measurable, as well as integrable in the organization's SDLC.
- There are several Maturity Models:
  - *Microsoft SDL Optimization Model*
  - *Building Security In Maturity Model* (BSIMM)
  - *Software Assurance Maturity Model* (SAMM)

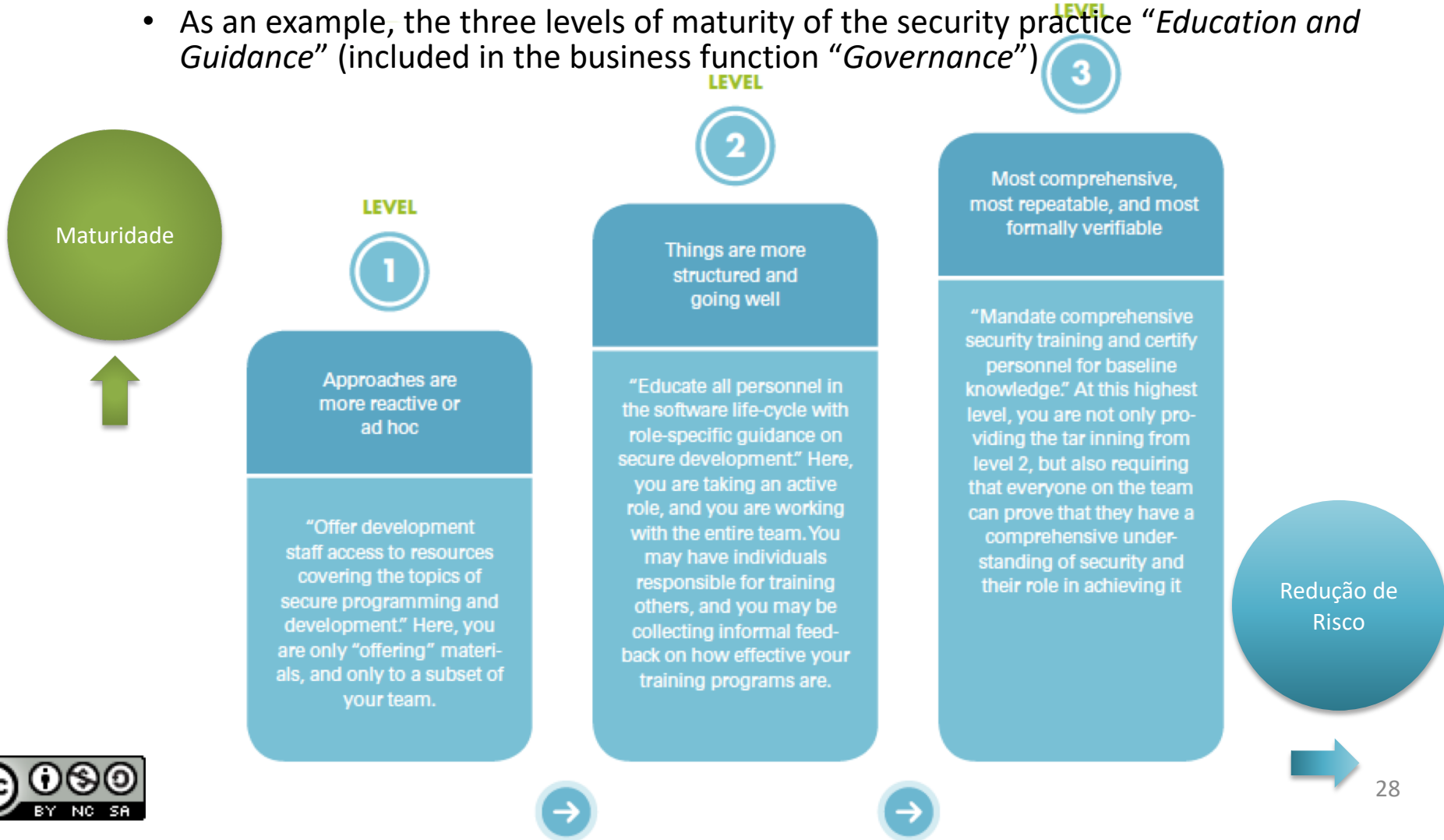
Secure Software Development Lifecycle (S-SDLC)

- *Software Assurance Maturity Model* (SAMM)
  - Purpose is to help organizations (whether developing, outsourcing or purchasing software) evaluate, formulate, and implement a software security strategy that can be integrated into the SDLC used.
  - Description of the Maturity Model:
    - Based on 12 **security practices**, which are grouped into 4 **business functions**;
    - Each security practice contains a set of activities, structured into three **levels of maturity**;
    - Activities at a lower level of maturity are more easily enforceable and require less formalization than activities at a higher maturity level.



Secure Software Development Lifecycle (S-SDLC)

- *Software Assurance Maturity Model* (SAMM)
  - Description of the Maturity Model:
    - As an example, the three levels of maturity of the security practice "*Education and Guidance*" (included in the business function "*Governance*")



Secure Software Development Lifecycle (S-SDLC)

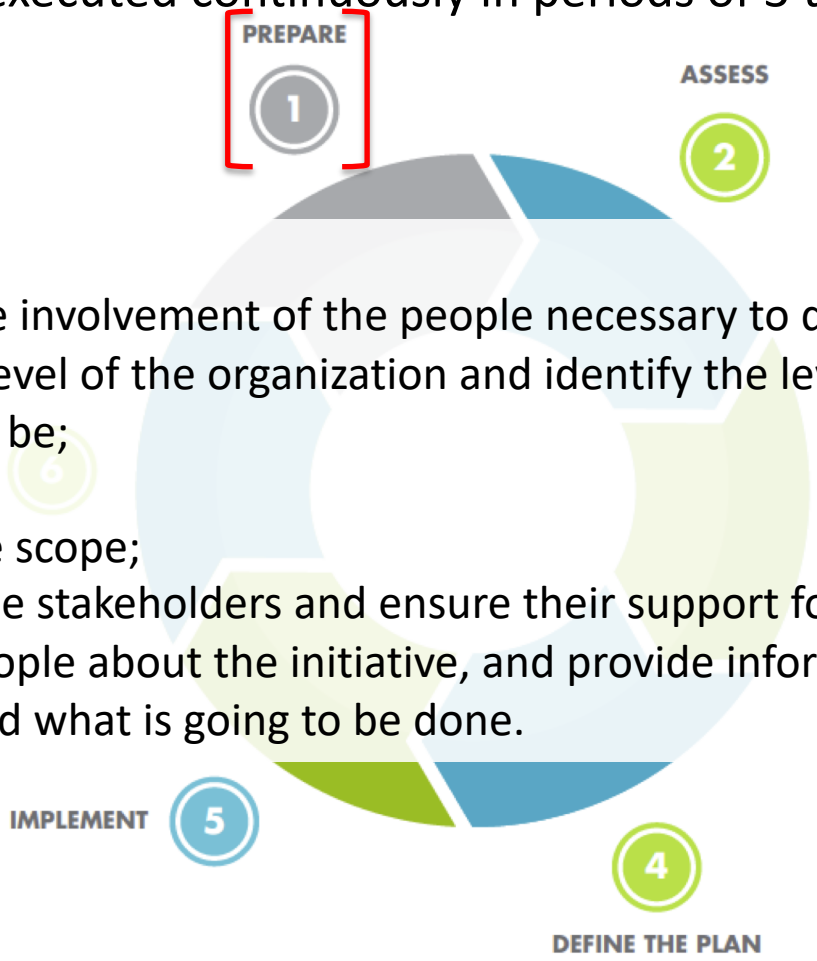
- *Software Assurance Maturity Model* (SAMM)
  - SAMM's value lies in providing a mechanism to know where the organization is on its path toward software security and, to allow to understand what is recommended to move to a higher maturity level.
  - SAMM does not insist that all organizations reach maturity level 3 in all categories. Each organization determines the maturity level of each ”Security Practice" that best fits their needs.
  - The configuration and cycle of the SAMM maturity model are made to allow:
    i. the evaluation of current software security practices,
    ii. the definition of the goal that best fits the organization,
    iii. the formulation of the roadmap to achieve the defined objective, and
    iv. the implementation of specific activities of the security practices, following descriptive measures.

Secure Software Development Lifecycle (S-SDLC)

- *Software Assurance Maturity Model* (SAMM)
  - SAMM cycle, suitable for continuous improvement (in which case the cycle should be executed continuously in periods of 3 to 12 months)

Secure Software Development Lifecycle (S-SDLC)

- *Software Assurance Maturity Model* (SAMM)
  - SAMM cycle, suitable for continuous improvement (in which case the cycle should be executed continuously in periods of 3 to 12 months)



Goal:

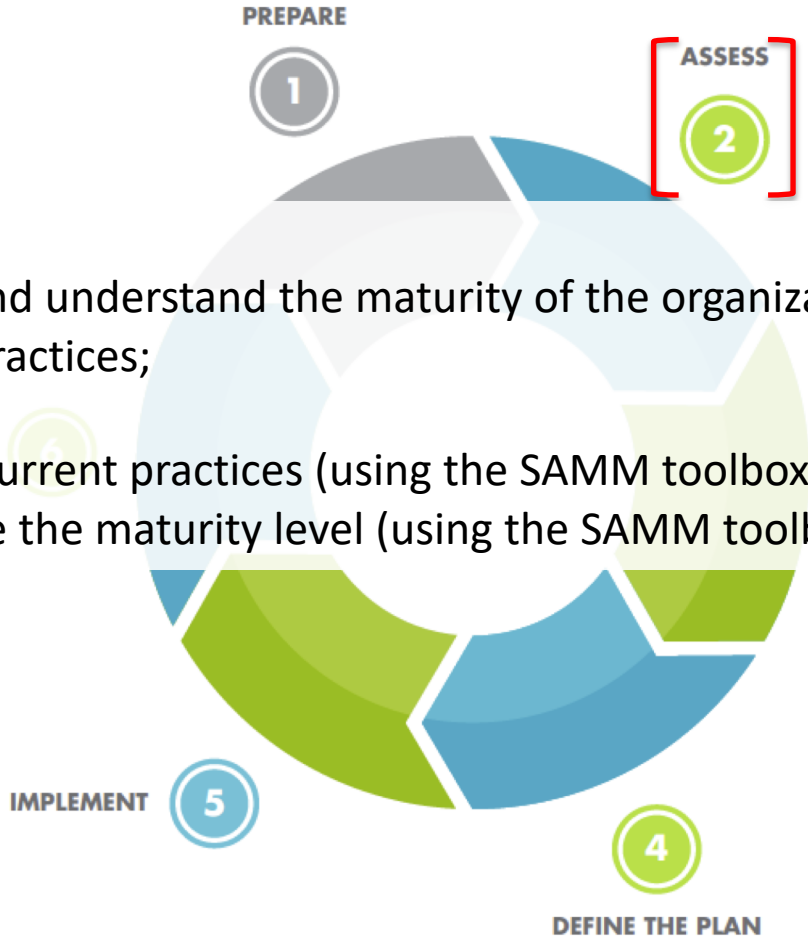
- Ensure the involvement of the people necessary to determine the current maturity level of the organization and identify the level of maturity in which it intends to be;

Activities:

- Define the scope;
- Identify the stakeholders and ensure their support for the project;
- Inform people about the initiative, and provide information that allows to understand what is going to be done.

Secure Software Development Lifecycle (S-SDLC)

- *Software Assurance Maturity Model* (SAMM)
  - SAMM cycle, suitable for continuous improvement (in which case the cycle should be executed continuously in periods of 3 to 12 months)



Goal:

- Identify and understand the maturity of the organization in each of the 12 security practices;

Activities:

- Evaluate current practices (using the SAMM toolbox);
- Determine the maturity level (using the SAMM toolbox).

Secure Software Development Lifecycle (S-SDLC)

- *Software Assurance Maturity Model* (SAMM)
  - SAMM cycle, suitable for continuous improvement (in which case the cycle should be executed continuously in periods of 3 to 12 months)

Goal:

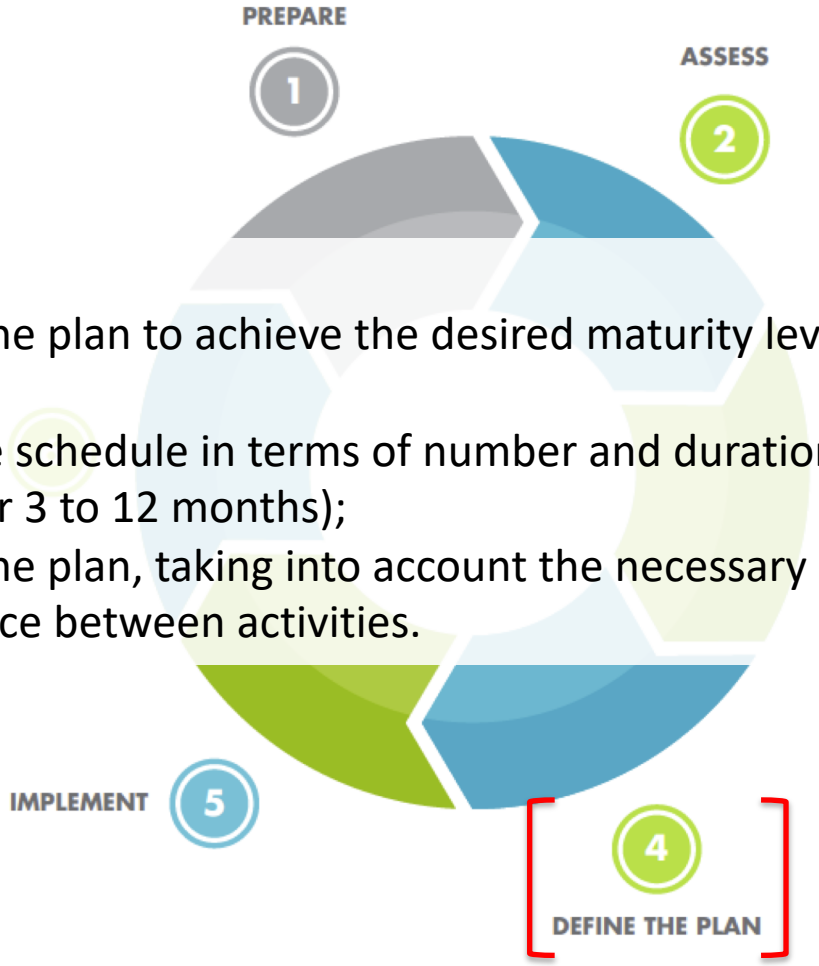
- Identify an objective score for each of the 12 safety practices, which serves as a guide to act on the most important activities;

Activities:

- Define the goal (using the SAMM toolbox);
- Estimate the impact of the goal on the organization (in financial terms, if possible).

Secure Software Development Lifecycle (S-SDLC)

- *Software Assurance Maturity Model* (SAMM)
  - SAMM cycle, suitable for continuous improvement (in which case the cycle should be executed continuously in periods of 3 to 12 months)



Goal:

- Develop the plan to achieve the desired maturity level (identified in phase 3);

Activities:

- Define the schedule in terms of number and duration of phases (usually 4 to 6 phases, for 3 to 12 months);
- Develop the plan, taking into account the necessary effort and possible dependence between activities.

Secure Software Development Lifecycle (S-SDLC)

- *Software Assurance Maturity Model* (SAMM)
  - SAMM cycle, suitable for continuous improvement (in which case the cycle should be executed continuously in periods of 3 to 12 months)

Goal:

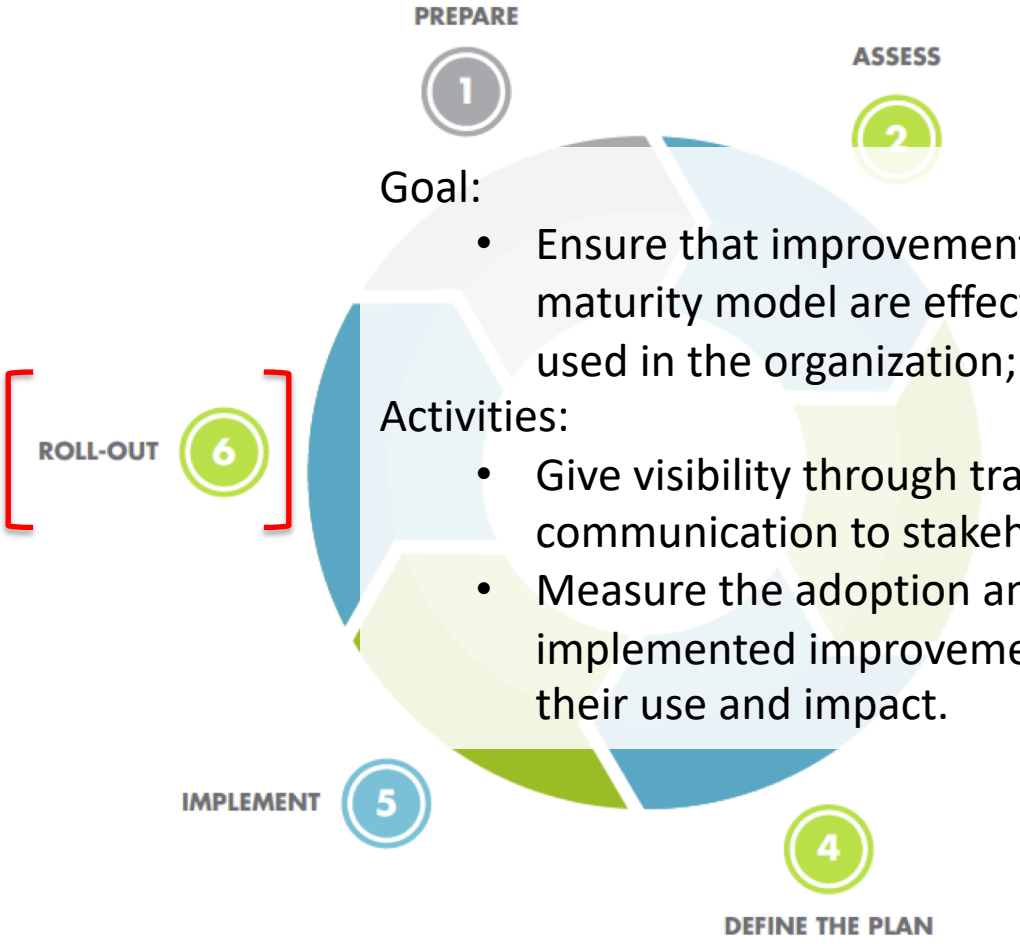
- Implementation of the plan to achieve the desired maturity level (defined in phase 4);

Activities:

- Planned implementation activities, taking into account their impact on processes, people, knowledge, and tools.

Secure Software Development Lifecycle (S-SDLC)

- *Software Assurance Maturity Model* (SAMM)
  - SAMM cycle, suitable for continuous improvement (in which case the cycle should be executed continuously in periods of 3 to 12 months)



Goal:

- Ensure that improvements in the software security maturity model are effective and are being followed / used in the organization;

Activities:

- Give visibility through training sessions and communication to stakeholders.
- Measure the adoption and effectiveness of the implemented improvements, through analysis of their use and impact.