ESOF

Termo 'software engineering' surgiu em 1968 e 1969 em duas conferências sobre engenharia de software feitas pela NATO.

Slides Index

  - Details and **techniques**: P6-7
  - Useful terms (+examples): P8-9
  - How to write: P10-11
  - Why use them: P12-14
- [9] **Architecture & design**
  - Architecture in **traditional** engineering: P23
  - Architecture in **software** engineering: Difficulties – P24; Role – P45
  - Software architecture: P27-28
  - **Reusing**: P29
  - Architectural **styles & patterns**: P30-40
  - **Component Diagrams**: P41-47
  - **Deployment Diagrams**: 48-51
  - **Package Diagrams**: P52-53
- [10] **Flutter**
- [11] **Behavior Driven Development**
  - What it is/does: P9
  - Gherkin: P10
- [12] **OpenCX**
  - What it is/does: P2
- [13] **Project Management:**
  - The Iron Triangle (**quality**): P14
  - Controlling Software Projects (resources, time, scope, quality): P15
  - **Resource** variable: P16
  - **Time** variable: P17
  - **Scope** variable: P18
  - Differences on **managing Agile processes**: P19
  - **Iterative and Incremental** (management): P20
  - **Paralell and Concurrent** Activities: P21
  - **Predictive vs. Agile Planning**: P22
  - Project **Balance in an Agile Processes**: P23
  - **Heroic vs. Collaborative**: P24
  - Management by **Facilitation**: P25
  - Iteration 0: P26
  - Iteration 1-N: P27
  - Three **Types of Iteration Plans**: P28
  - Next Iteration Plan: P29
  - **Estimation**: P30
  - Scope Evolution: P31
- [14] **Construction Evolution**
  - Why it's important: P6-8
  - **Evolution** and servicing (and **phase-out**): P9-10
  - Evolution processes: P11-16
  - **Types of maintenance**: P17-19
  - Maintenance prediction: P21
  - Re-engineering process activities: P22-24
- [15] **History of floss**
  - Copyright vs. Copyleft (GPL): P12
  - 5 Principles of Open Source Way: P48
- [16] **Open Source Labor economics**
  - Career path: P10
  - Values and signals: P12
  - Value to developer: P21
  - Become a commiter: P22

Software reuse

- **Abstraction level** - Knowledge of successful abstractions is reused.
- **Component level** - Collections of objects are reused.
- **Design/System level** - The entire application design/systems are reused.
- **Object level** - Objects, classes and methods from libraries are reused.

Object level

Requirements Engineering Activities

Conference paper "Maintenance, Maintainability, and System Requirements Engineering", 1964. Came into general use in 1990s with the publication of an IEEE Computer Society tutorial.

Inception

- The requirement engineer asks a set of questions to establish a software process.
- He understands the problem and evaluates the proper solution.
- He establishes the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the questions.

Elicitation

Elicitation means to find the requirements from anybody: interviews, prototypes, user stories, etc…
Difficult because:

- Problem of scope: The customer gives unnecessary technical detail rather than the overall system objective.
- Problem of understanding: Poor understanding between the customer and the developer regarding various aspect of the project.
- Problem of volatility: The requirements change from time to time.

Elaboration

- The information taken from the user during inception and elaboration are expanded and refined.
- Develop a pure model of software using functions, features and constraints of a software.

Negotiation

The software engineer decides how the project will be achieved with limited business resources. Estimate the impact of the requirement on the project cost and delivery time.

Specification

The requirement engineer constructs a final work product. Formalization of the proposed software: informative, functional and behavioral. The formalization can be in both graphical and textual formats.

Validation

The work product is built as an output of the requirements and its quality is accessed through a validation step. The formal reviews from the software engineer, customer and other stakeholders help the primary requirements' validation mechanism.

Requirement management

Set of activities that help the project team identify, control and track the requirements and changes to be made to the requirements at any time of the ongoing project.
These activities start with the identification and assigning of a unique identifier to each of the requirements (requirement traceability table).

UML Diagrams

Most used: Use-Case, Class, State machine, Communication, Sequence, Component, and Deployment.

Structural

Class Diagram Class diagrams are the main building block of any object-oriented solution. It shows the classes in a system, attributes, and operations of each class and the relationship between each class.

Component Diagram A component diagram displays the structural relationship of components of a software system. These are mostly used when working with complex systems with many components. Components communicate with each other using interfaces. The interfaces are linked using connectors.

Deployment Diagram A deployment diagram shows the hardware of your system and the software in that hardware. Deployment diagrams are useful when your software solution is deployed across multiple machines with each having a unique configuration.

Object Diagram/Instance diagrams Object Diagrams are very similar to class diagrams. They also show the relationship between objects, but they use real-world examples.
They show how a system will look like at a given time.

Package Diagram Shows the dependencies between different packages in a system.

Profile Diagram This is a new type of diagram that is very rarely used in any specification.

Composite Structure Diagram Composite structure diagrams are used to show the internal structure of a class.

Behavioral Diagrams

Use Case Diagram These give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.
It's a great starting point for project discussion, because you can easily identify the main actors involved and the main processes of the system.

Activity Diagram Activity diagrams represent workflows in a graphical way. They can be used to describe the business workflow or the operational workflow of any component in a system. Sometimes used as an alternative to State machine diagrams.

State Machine/State chart/State Diagram Similar to **activity diagrams**. These describe the behavior of objects that act differently according to the state they are in at the moment.

Sequence Diagram Show how objects interact with each other and the order those interactions occur. They show the interactions for a particular scenario. The processes are represented vertically and interactions are shown as arrows.

Communication/Collaboration Diagram Similar to sequence diagrams, but the focus is on messages passed between objects. The same information can be represented using a sequence diagram and different objects.

Interaction Overview Diagram Similar to activity diagrams. **Activity** diagrams show a sequence of processes, **interaction overview** diagrams show a sequence of interaction diagrams (collection of interaction diagrams).

Timing Diagram Similar to sequence diagrams. They represent the behavior of objects in a given time frame. If there is more than one object is involved, this diagram is used to show interactions between objects during that time frame.

Open-Source software

Open-Source software can be built and sold individually. Free software is a subset of open-source software that can't be sold (GNU GPL license). Software that include other free-software also has to be licensed as free-software (viral license) and so, can't be sold.

Historic software bug

On June 4th, 1996, 30 seconds after the launch, the Ariane 5 rocket disintegrated and exploded. Afterwards, simulations with a similar flight system and conditions revealed that in the rocket's software (which came from Ariane 4), a 64-bit floating point variable [4.9x10^-307, 1.8x10^+308] was being interpreted as a 16-bit integer variable, [−32768, 32767]. This caused a series of problems affecting most computers and hardware on-board. After a few seconds, the entire ship paralyzed and self-destructed.