Software Engineering

Introduction

SoftEng
http://softeng.polito.it

Version 2.0

Outline

- Definitions and concepts
- Process and product
- Process and product properties
- Laws
- Principles

Definitions

Definition of Software Engineering

- **Multi person** construction of **multi version** software

- Multi person:
  - Issues in communication: misunderstandings, language gaps
  - Issues in coordination
- Multi version:
  - Issue in maintenance over many years

Definition of Software

- A collection of
  - Computer programs,
  - Procedures,
  - Rules,
  - Associated documentation
  - Data
- Example:
  - Requirements document,
  - Project plan,
  - Test plan,
  - Test cases,
  - Build scripts,
  - Deployment scripts,
  - User manuals

Software Types

- Embedded in non software product
  - Car, washing machine, ..
  - Production line (Industry 4.0)
- Stand alone
  - Office suite, social network, ..
- Embedded in enterprise
  - Information system

Software Criticality

- Safety critical: harms people or environment
  - Self driving car
  - (usually embedded software)
- Mission critical: harms business
  - Banking, finance, retail
  - (usually embedded in enterprise)
- Other

Beware

- Software is not free
- Software changes (and it is not easy to change)
- Software is not perfect
- Software is complex

Process and Product

SOftEng
http://softeng.polito.it

The Software “factory”

Classical Engineering

- Design the product
- Design the factory
- Manufacture the product
- Maintain the product

Classical Engineering		Software Engineering
• Design the product	→	• Software product
• Design the factory	→	• Software Process
• Manufacture the product	→	• Deployment and Delivery
• Maintain the product	→	• Evolution and Maintenance

Software Product

- Functional properties
  - Do this…
  - Do that…
  - …
  - Use Cases

Software Product properties

- Non functional properties
  - Usability
    - Effort needed to learn using the product (installation, day to day usage)
    - Satisfaction expressed by the user
    - Existence of functions needed by the user
  - Efficiency
    - For a given function in a given context: response time
    - For a given function / for a complete product:
      - Memory
      - CPU
      - Bandwidth
      - energy used

Software Product properties

- Non functional properties
  - Reliability / availability
    - Defects visible by end user per time period / Probability of defect over a time period
    - Percentage of time the product is / is not available to end user
  - Maintainability
    - Effort (person hours) needed to add /modify / cancel a software function
    - Effort to fix a defect
    - Effort to deploy on a different platform (DB, OS, ..)

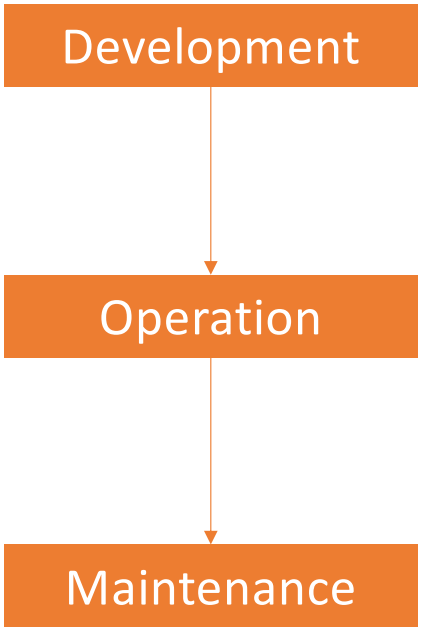
Software Product properties

- Non functional properties
  - Security
    - Protection from malicious access
    - Access only to authorized users
    - Sharing of data
  - Safety
    - Absence of harm to persons
    - Absence of hazardous situations for persons
  - Dependability
    - Safety + security + reliability

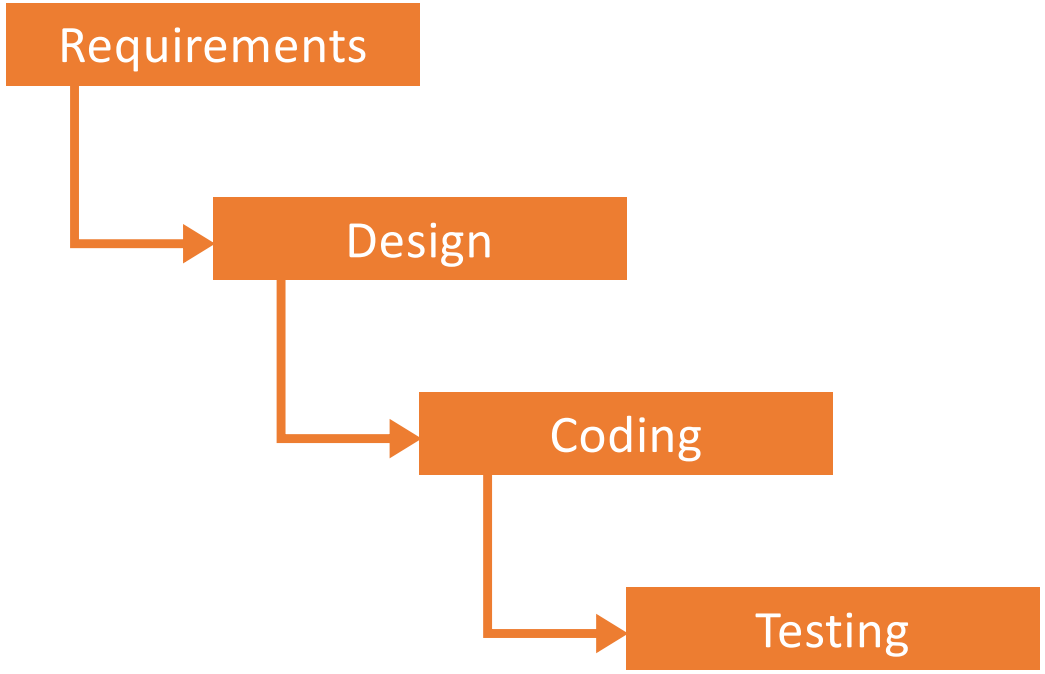
Software Product properties

- Non functional properties
  - Are difficult to engineer
  - Are often forgotten
  - Make the difference between competing products

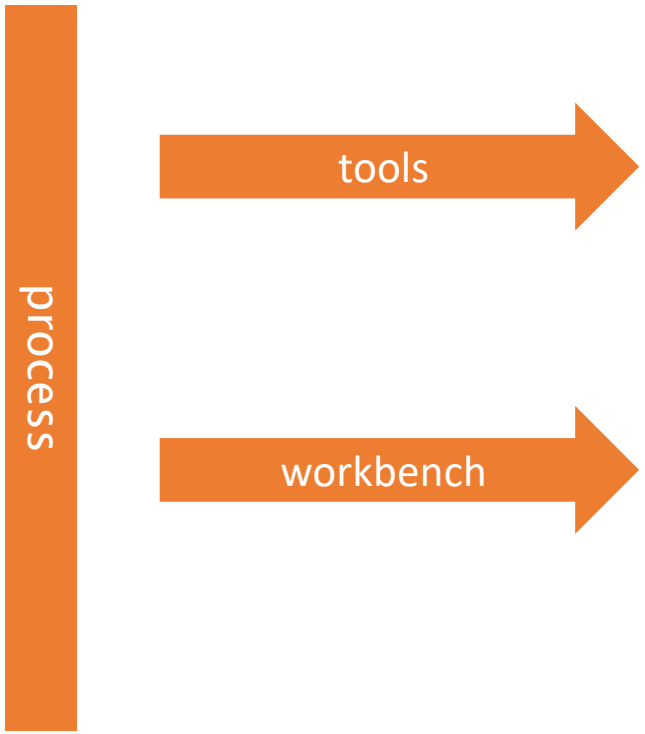
Software process



Software process – development phase



Classical Workbench



Software Workbench

- Git
- Subversion
- Polarion
- Clearcase
- Team Foundation Server

Software Tools

- Requirements
  - Context diagram
  - Stakeholders
  - Stories / Personas
  - Use cases
  - Scenarios
  - Functional requirements
  - Non functional requirements
  - Glossary
  - ....

Software Tools in this course

slack

GitLab

CATME
SMARTER Teamwork

zoom

SoftEng
http://softeng.polito.it

Software Tools

- Design
  - Component diagram
  - Package diagram
  - Class diagram
  - Interaction diagram
  - ....

Software Tools

- Development
  - Visual Studio Code
  - Intellij-Idea
  - Eclipse
  - Android Studio
  - Xcode
  - PyCharm
  - ….

Software Tools

- Test
  - Unit test, white box
  - Unit test, black box
  - ….

Process Properties

Process Properties

- Cost
  - Currency (€, $, …)
- Effort
  - Person hours
- Punctuality
  - Promised delivery date vs actual delivery date
- Conformance (to standards, norms)

Laws

Software Engineering Laws

[Endres Rombach 2005]

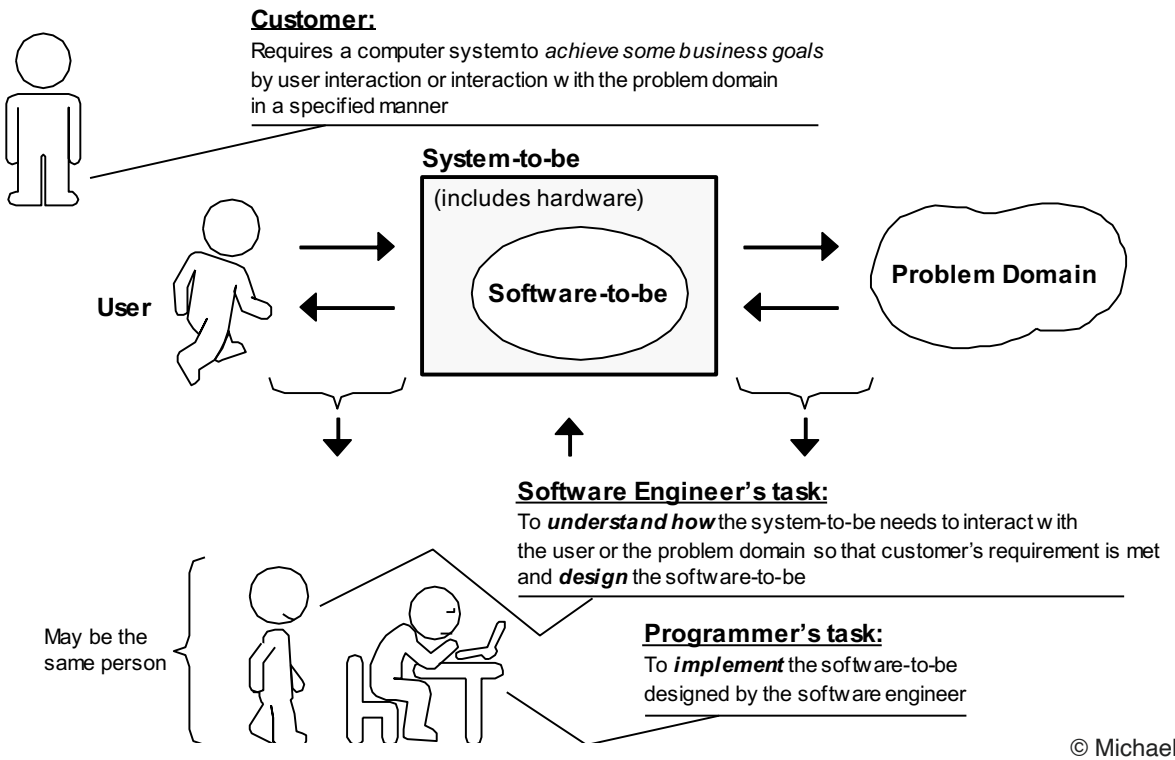
- Requirements deficiencies are the prime source of project failures
- Requirements and design cause the majority of defects
- Defects from requirements and design are the more expensive to fix

Software Engineering Laws

- Modularity, hierarchical structures allow to manage complexity
- Reuse guarantees higher quality and lower cost

Software engineering Laws

- Good designs require deep application domain knowledge



Software Engineering Laws

- Testing can show the presence of defects, not their absence
- A developer is unsuited to test his/her code

Software Engineering Laws

- A system that is used will be changed

- An evolving system will increase its complexity, unless work is done to reduce it
  - Architecture erosion
  - Requirements creep
  - Refactoring

Software Engineering Laws

- Developer productivity varies considerably
- Development effort is a (more than linear) function of size
- Adding resources to a late project makes it later

Software Engineering Laws

- The process should be adapted to the project

Information System laws

- Conway's law
- Structure of a system produced by an organization mirrors the communication structure of the organization
  - Applied to organizational structure: the structure should mirror how members work together
  - Applied to information systems: IS parts will reflect the organizational structure
  - Applied to software projects: module interfaces will reflect the teams' structure

Principles

KISS

Never add

accidental complexity

to

essential complexity

Accidental complexity

“Kindly extend your hand in my direction, bearing in mind the physical properties of the object commonly referred to as "salt," which is typically utilized as a seasoning agent to enhance the flavor profile of comestibles, and transmit it to my vicinity with a controlled force and trajectory so as to facilitate its transfer to my immediate possession, thus allowing for its incorporation into the culinary creation currently under consideration.”

Essential complexity

Pass me the salt, please

“Complexity is your enemy.
Any fool can make something complicated.
It is hard to keep things simple.”

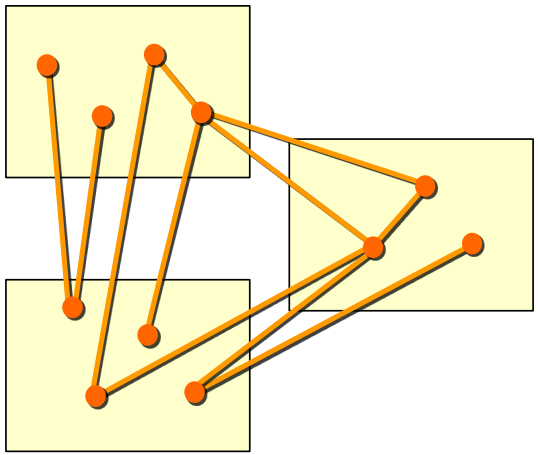
[Richard Branson]

Separation of concerns

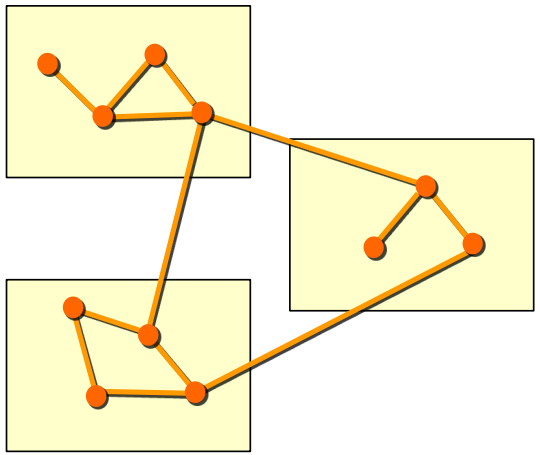
- Given a large, difficult problem, try to split it in many (independent) parts, and consider a part at a time:
  - In war: divide and conquer
  - In SE: software process, concentrate on what the system should do, then on how, then do it
  - In SE: programming languages, separate error handling and error generation
  - In SE: divide complex system in (independent) components (modules)

Separation of concerns

- Divide a complex system in modules, with high cohesion and low coupling



high coupling



low coupling

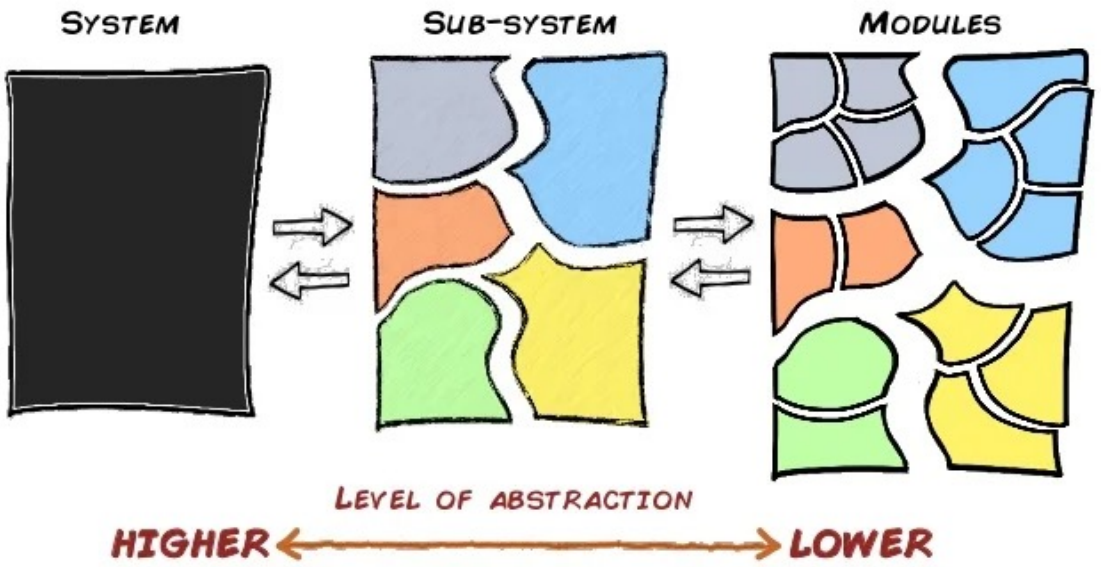
Separation of concerns

Examples

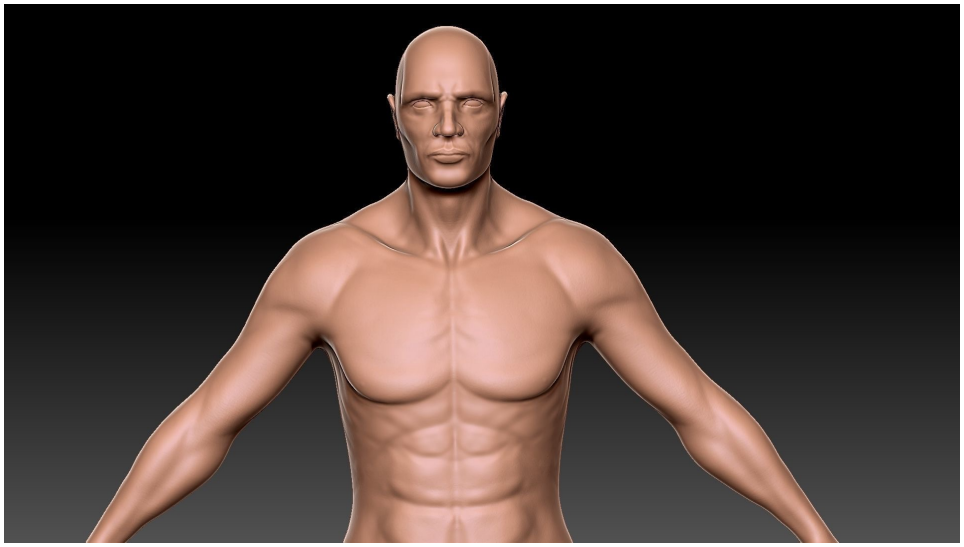
- ISO OSI communication stack, 7 layers
- 3 layers architecture (data base, application logic, presentation)
- Model View Controller

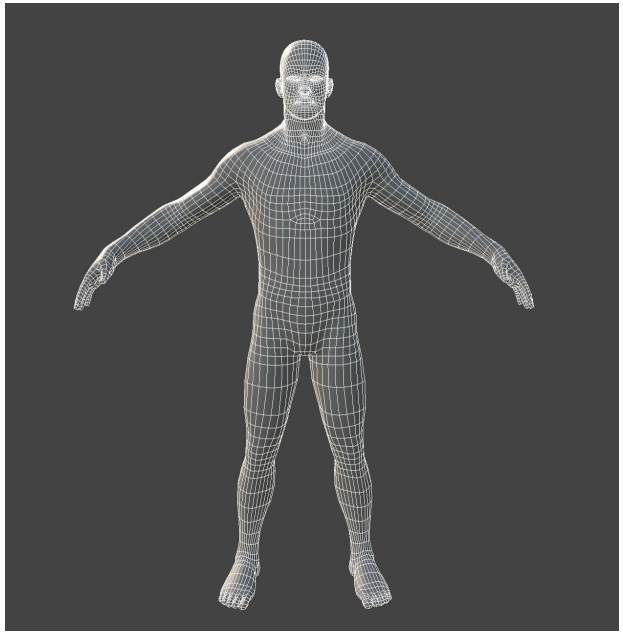
Abstraction

- Given a difficult problem or system, extract a simpler view of it, avoiding unneeded details, then reason on the abstract view (model)



marcello.thiry@gmail.com





Recap

- Software is not only code
- Software process, software products
- Properties
- Laws of Software Engineering and Information Systems
- Principles to be followed