Introduction to Software Engineering

Davide Falessi

Agenda

- Introduction to Software Engineering
  - Definitions
  - Practices
  - Process models

What is software engineering?

- Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software (IEEE, 1990).
  - AKA **software process model** or **software development process**.
- Software engineering is concerned with **all aspects of software production**

Software Development Practices

- *Software development practice: a disciplined, uniform approach to the software development process* (IEEE, 1990).
  - A well-defined activity that contributes towards the satisfaction of the project goals;
  - Generally the output of one practice becomes the input for the next practice.

List of practices (incomplete)

- Requirements engineering
- System analysis
- High-level design/architecture
- Low-level design
- Coding
- Integration
- Design and code reviews
- Testing
- Maintenance
- Project management
- Configuration management

Best Practice

- A **best practice** is a method that has shown results **superior** to those achieved with other means, and that is used as a benchmark.
- In addition, a "best" practice can evolve to become better as improvements are discovered.
- Best practice is considered by some as a business buzzword, used to describe the process of developing and following a **standard (i.e., very common) way** of doing things that several organizations are using.

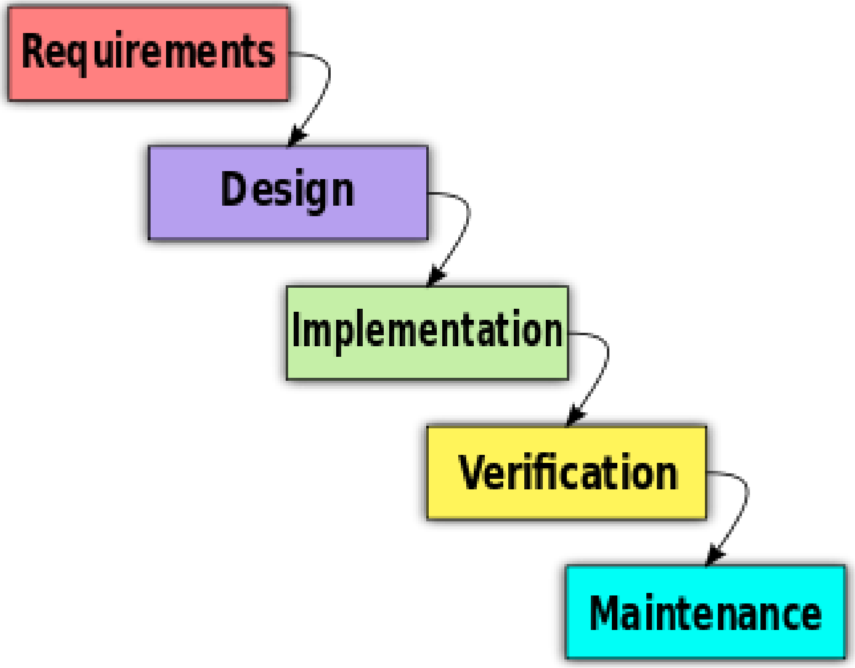
Software development process

- A software development **process** is the set of activities (or practices), their **order**, and **importance**.
- The process can be structured in phases, and each phases can apply all activities, with different emphasis across phases.

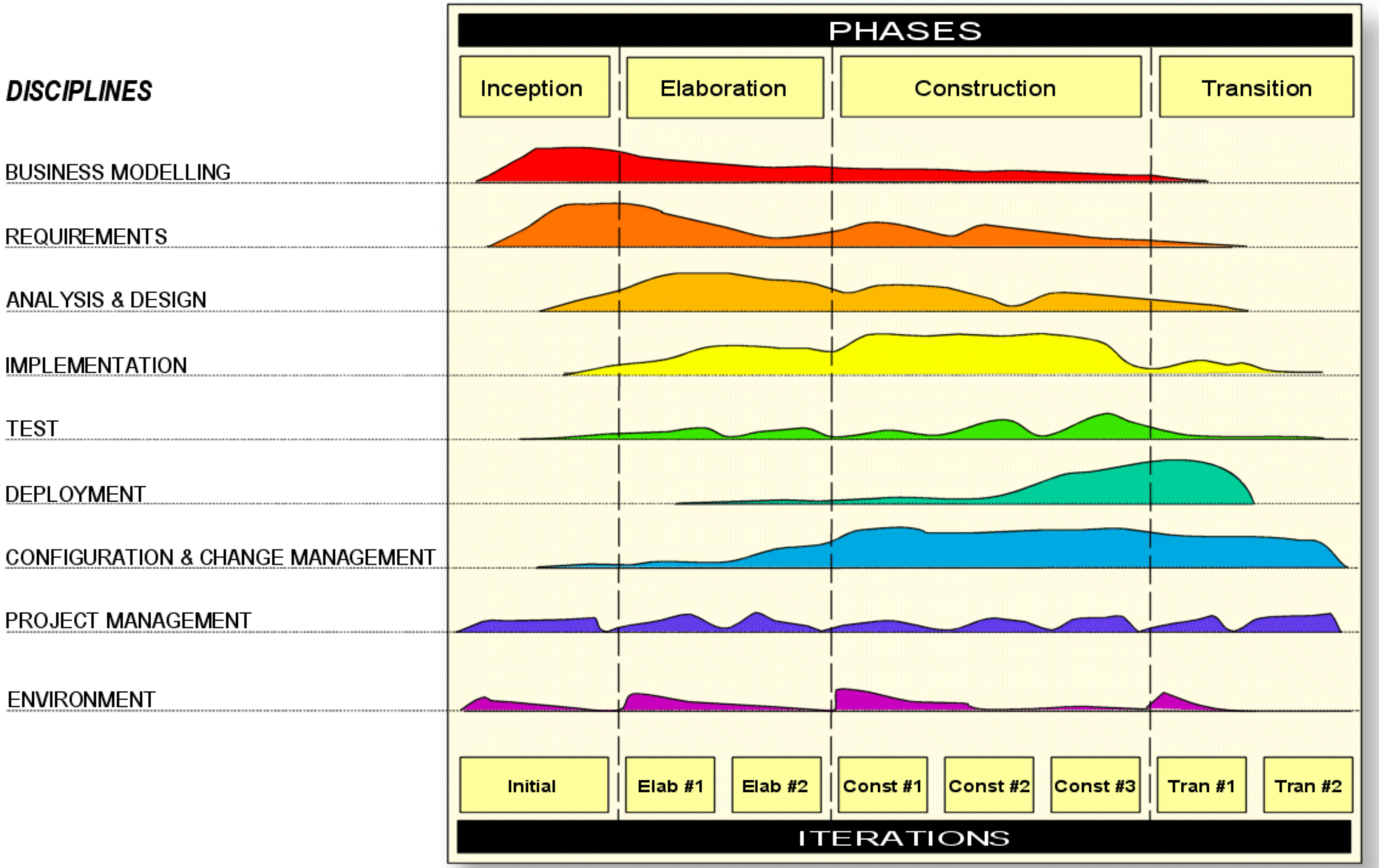
Process Model

- A software process model is a *simplified, abstracted description of a software development process*.
- The primary purpose of a software process model is to determine the order of phases involved in software development and to establish the transition criteria for progressing from one phase to the next (Boehm, May 1988).
- Because of the simplification, several software development methodologies may share one process model – the differentiation is in the details of the process itself.

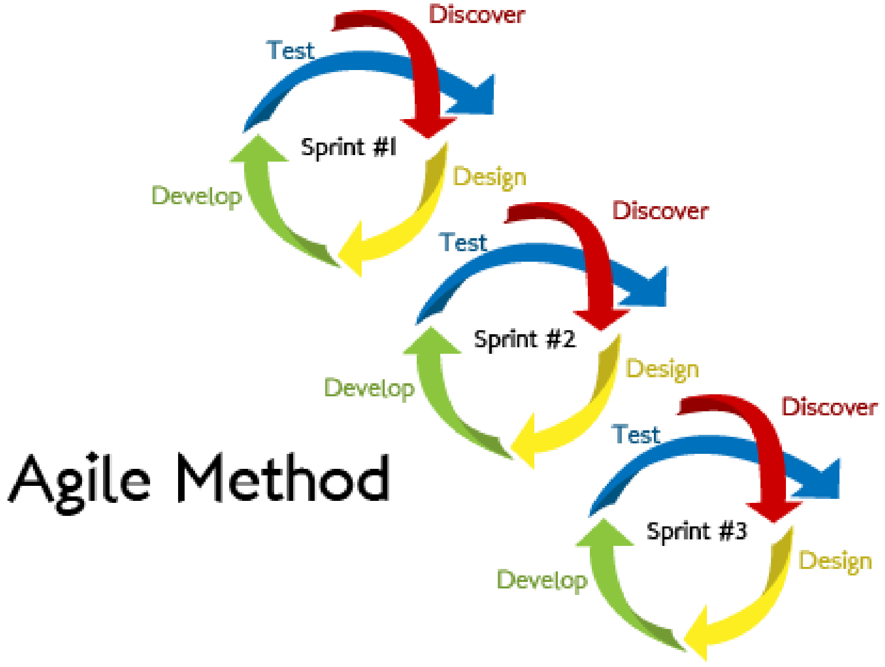
Waterfall Model



RUP


PHASES
DISCIPLINES
Inception
Elaboration
Construction
Transition
BUSINESS MODELLING
REQUIREMENTS
ANALYSIS & DESIGN
IMPLEMENTATION
TEST
DEPLOYMENT
CONFIGURATION & CHANGE MANAGEMENT
PROJECT MANAGEMENT
ENVIRONMENT
Initial
Elab #1
Elab #2
Const #1
Const #2
Const #3
Tran #1
Tran #2
ITERATIONS

Agile Model


Discover
Test
Sprint #1
Design
Develop
Discover
Test
Sprint #2
Design
Develop
Discover
Test
Sprint #3
Design
Develop
Agile Method

Plan-driven or Agile?

- Recently, process models have begun to be characterized as plan-driven or agile (Boehm, 2002).
- Plan-driven models: Maximize plan.
- Agile: Minimize plan (because it will change anyway).

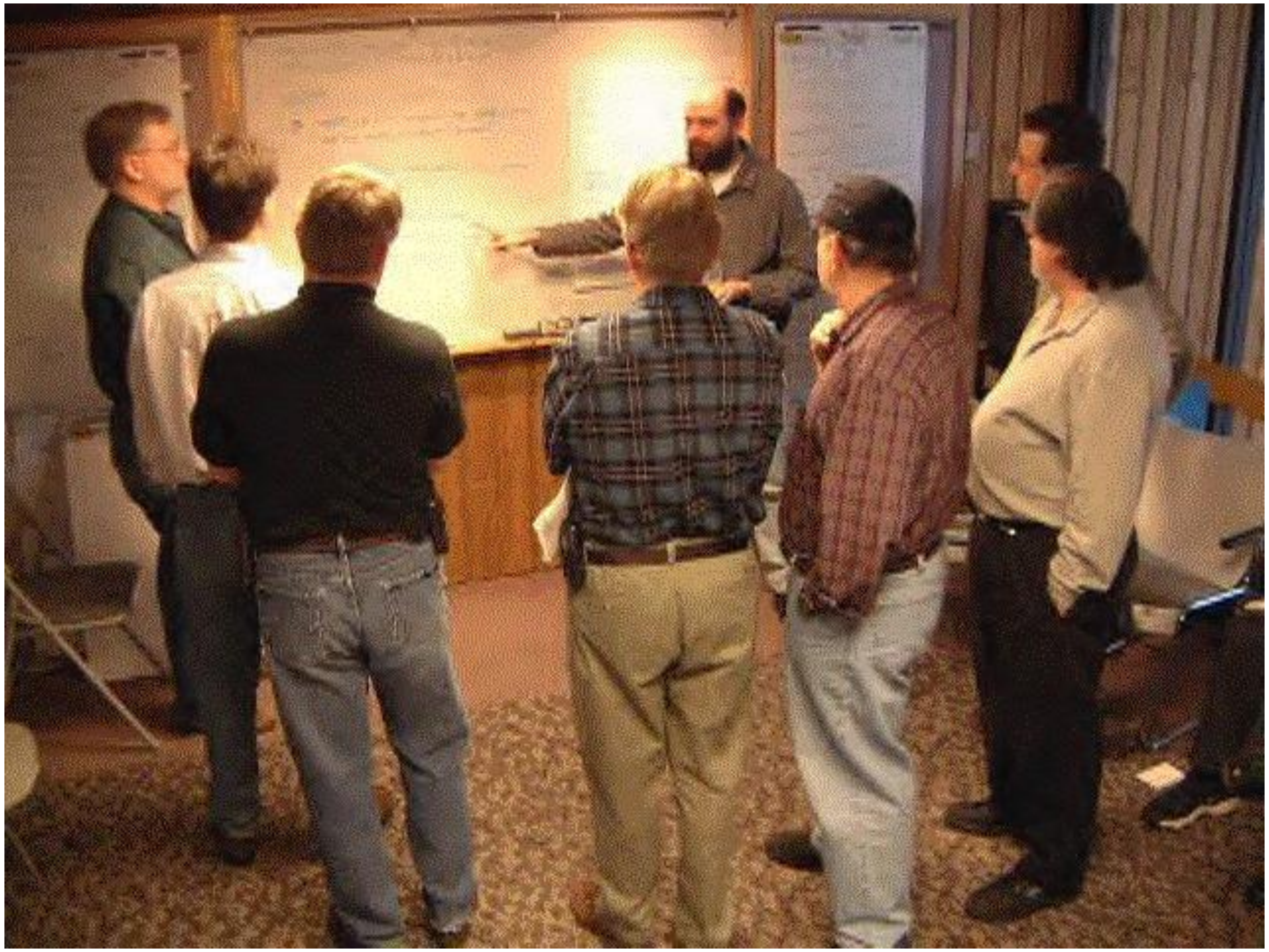
Plan-driven models

- Can be summarized as "Do it right the first time."
- Are very appropriate for projects in which there is not a great deal of requirements and/or technology **changes** anticipated throughout the development cycle.
- Suitable for safety- and mission-critical systems because of their emphasis on defect prevention and elimination.(Boehm, 2002)
- Examples: Personal Software Process (Humphrey, 1995), the Rational Unified Process (Jacobson, Booch et al., 1999), and Cleanroom Software Engineering (Mills, Linger et al., 1987).

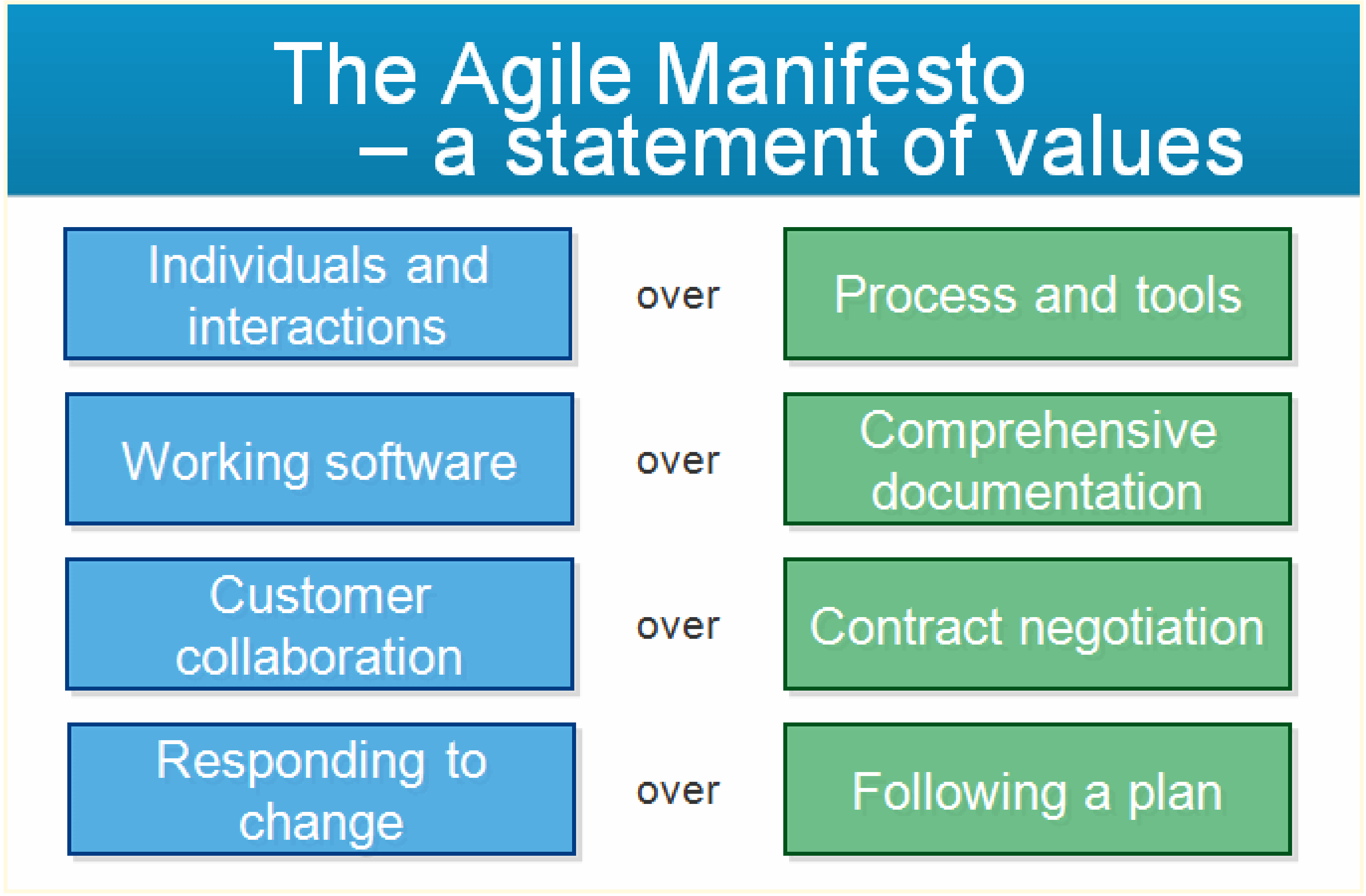
Agile models

- Spending a limited amount of time on planning and requirements gathering early in the process and much more time planning and gathering requirements for small iterations **throughout the entire lifecycle of the project**.
  - I.e., the main difference between plan-driven and agile process is not in the amount of effort spent in requirement, but on the **when** this effort is spent.
- Better suited for projects in which a great deal of **change** is anticipated. (Boehm, 2002)
- Examples: Extreme Programming (XP) (Beck, 2000), Scrum (Schwaber and Beedle, 2002), Crystal (Cockburn, 2001), FDD (Coad, LeFebvre et al., 1999), and DSDM (Stapleton, 1997).

Agile manifesto meeting



Agile Manifesto



The Agile Manifesto – a statement of values

Individuals and interactions	over	Process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

Wrong Hopes: 1 Requirements do not change

- It is often very hard for customers to express exactly what they want in a product (software is only thought-stuff for them too!). They often don’t know what they want until they see *some* of what they’ve asked for.
- Requirements analysts may not understand the product domain as completely as they need to early in the product lifecycle.
- Product domain can be constantly changing during the course of a product development cycle. New technology becomes available. Competitors release new products that have features that weren’t thought of. Innovators think of wonderful new ideas that will make the product more competitive.

Wrong Hopes: 2 The project will go as planned

- Software engineers are an **optimistic** crew (natural selection?)
- “Next time, things will go more smoothly. We know so much more now.”
  - “never on time” reputation.

Stakeholders

- Stakeholders are every person involved in the development project. These include:
  - ***end users***
  - ***customers***
  - ***domain experts***
  - ***analysts***
  - ***implementers***
  - ***testers***
  - ***managers***
  - ***visionaries***
  - ***maintainers and operators***
  - ***Any other interested parties***

Problems vs Solutions

- Viewing process as problem solving:
  - Requirements & specification are
    - ***problem statement***
  - Design & implementation are
    - ***problem solution***