Software Engineering

Object Oriented Programming

http://softeng.polito.it/courses/09CBI

SOftEng
http://softeng.polito.it

Version 0.4.0 - March 2018

© Maurizio Morisio, Marco Torchiano, 2018

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.
You are free: to copy, distribute, display, and perform the work

Under the following conditions:


Attribution. You must attribute the work in the manner specified by the author or licensor.


Non-commercial. You may not use this work for commercial purposes.


No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

Software Engineering

- The origin of the discipline
  - Garmish 1968
  - NATO organized conference
    - Motivation was that the computer industry at large was having a great deal of trouble in producing large and complex software systems

Software Engineering

Multi person construction of multi version software

- Parnas

SE

A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers

- Ghezzi, Jazayeri, Mandrioli

Software vs. Program

Software ≠ Program

- Software..
  - includes rules, documentation, data…
  - is long-lived
  - has many stakeholders
  - depends on several humans developers
  - is ~10 times more expensive

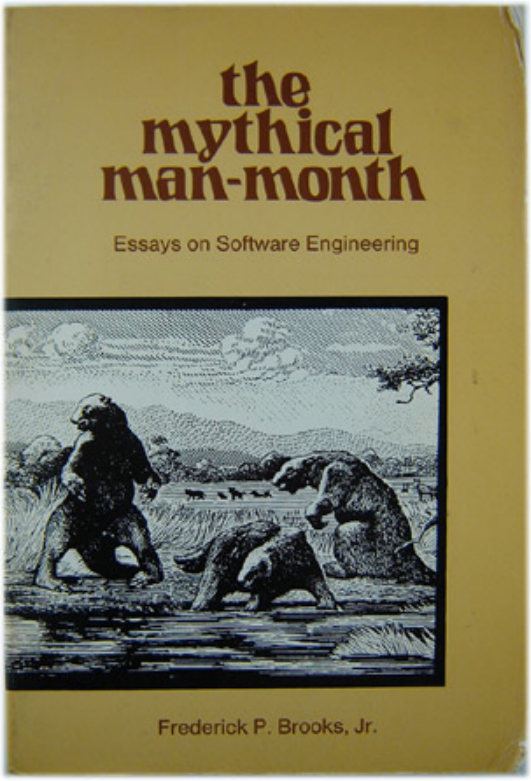
Software Discipline Premises

- Evolutionary and experimental
  - Software does not ages, its context does
- Development as opposed to production
  - Replication is almost free
- Makes use of technologies that are ultimately human based
  - Human issues are as important as technical ones

The mythical man-month

- Fred Brooks, 1975

Adding manpower to a late software project makes it later.



Software is Software is Software?

- No!
- All software is not the same
  - Process is a variable
  - Goals are variable
  - Content varies
  - …

SOFTWARE LIFE CYCLE

Goal

Produce software

- documents, data, code

with defined, predictable process properties

- cost, duration

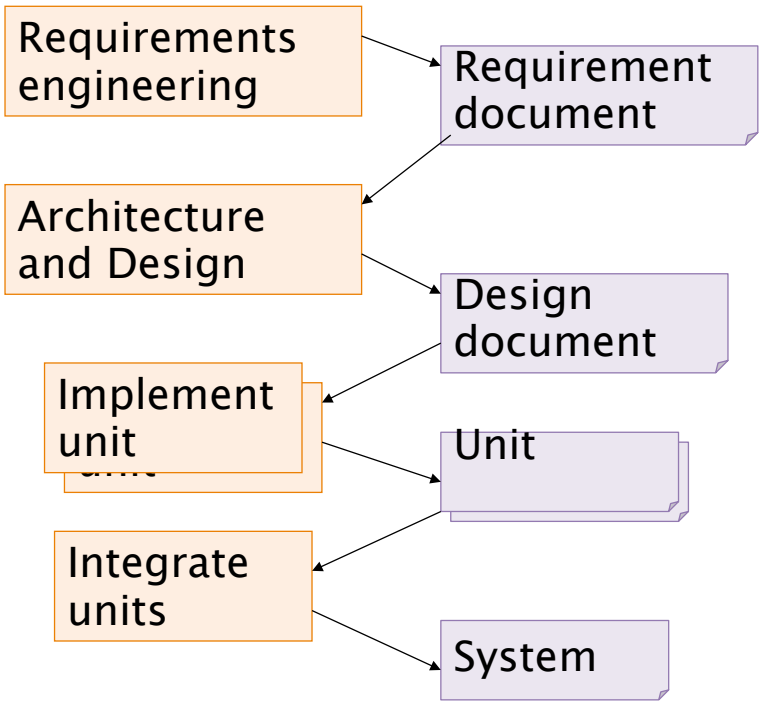
and product properties

- functionality, reliability, performance, ..

The production activities

- Requirement engineering
  - What the software should do
- Architecture and design
  - What units and how organized
- Implementation
  - Write source code
  - Integrate units

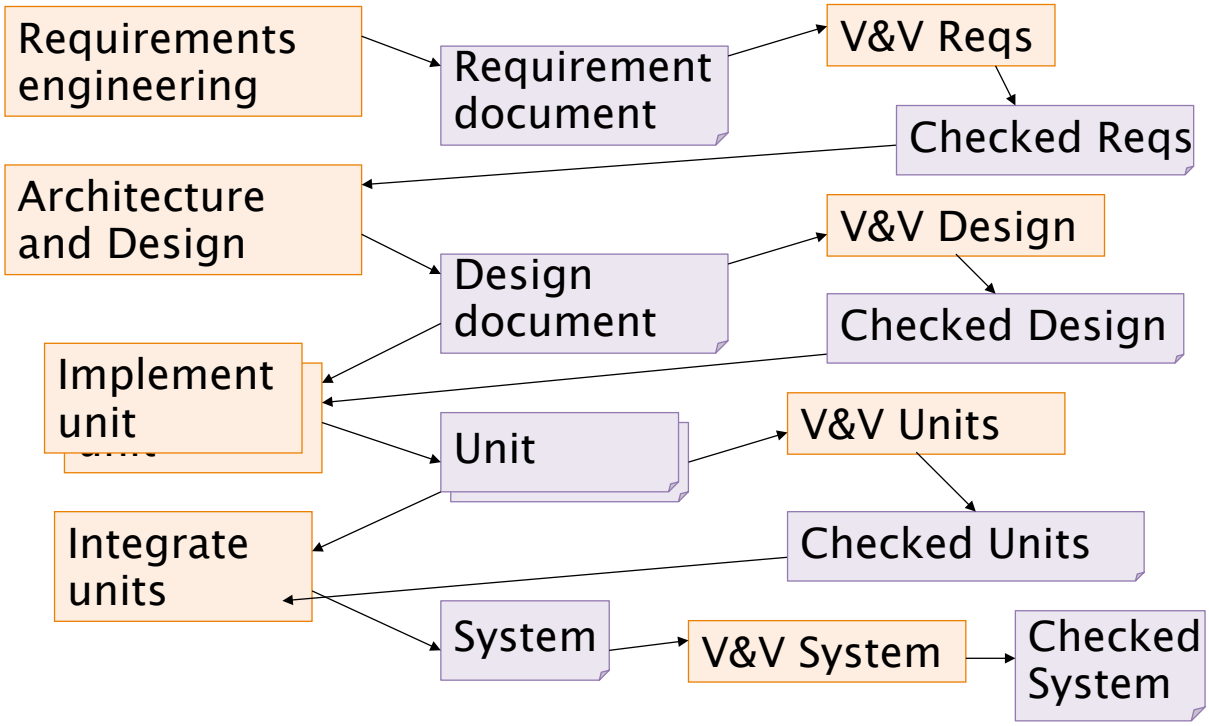
Production activities



The V & V activities

- V & V = verification and validation
- Control that the requirements are correct
  - Externally: did we understand what the customer/user wants?
  - Internally: is the document consistent?
- Control that the design is correct
  - Externally: is the design capable of supporting the requirements
  - Internally: is the design consistent?
- Control that the code is correct
  - Externally: is the code capable of supporting the requirements and the design?
  - Internally: is the code consistent (syntactic checks)

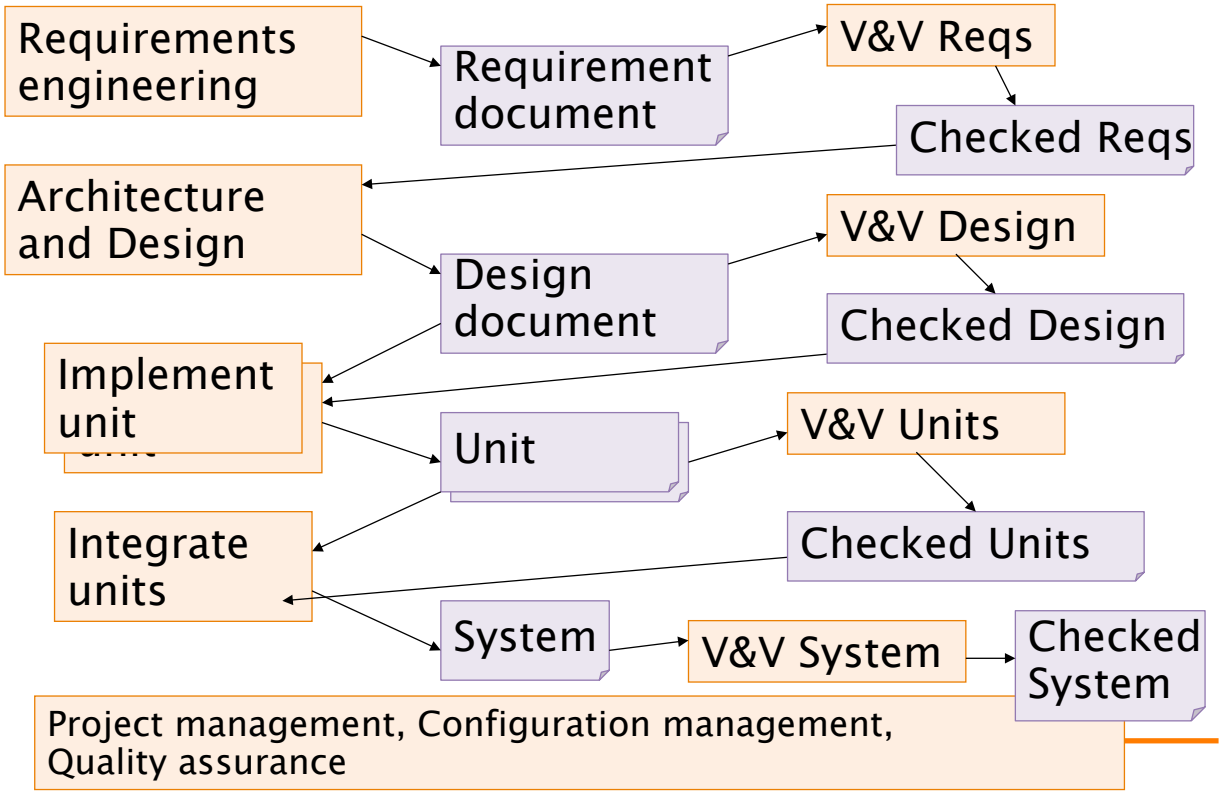
Production activities



The management activities

- Project management
  - Assign work and monitor progress
  - Estimate and control budget
- Configuration management
  - Identify, store documents and units
  - Keep track of relationships and history
- Quality assurance
  - Define quality goals
  - Define how work will be done
  - Control results

Production activities

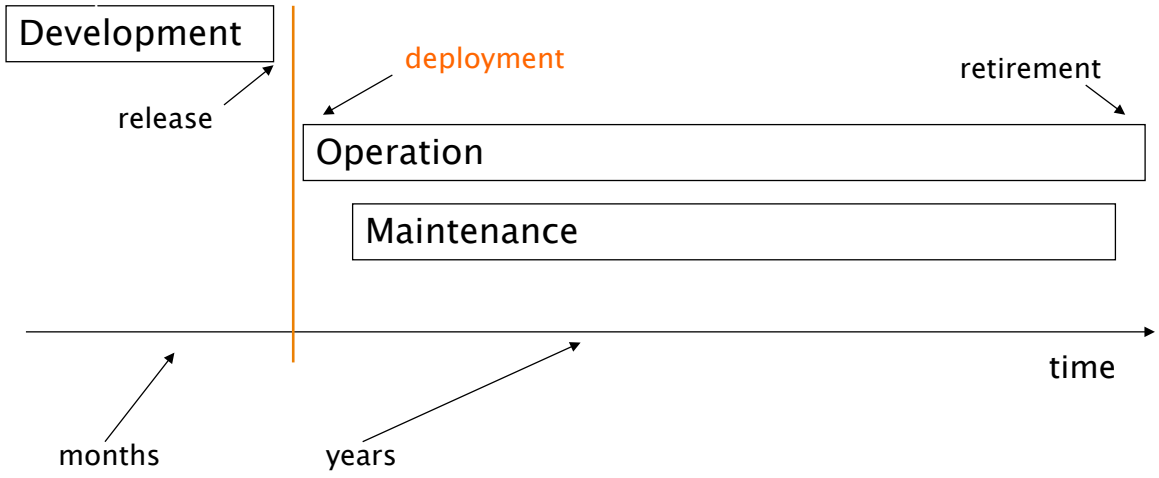


PHASES

Beyond development

- Development is only the first part of the game
  - Operate the software
    - Deployment
    - Operation
  - Modify the software
    - Maintenance
  - Terminate the usage
    - Retirement

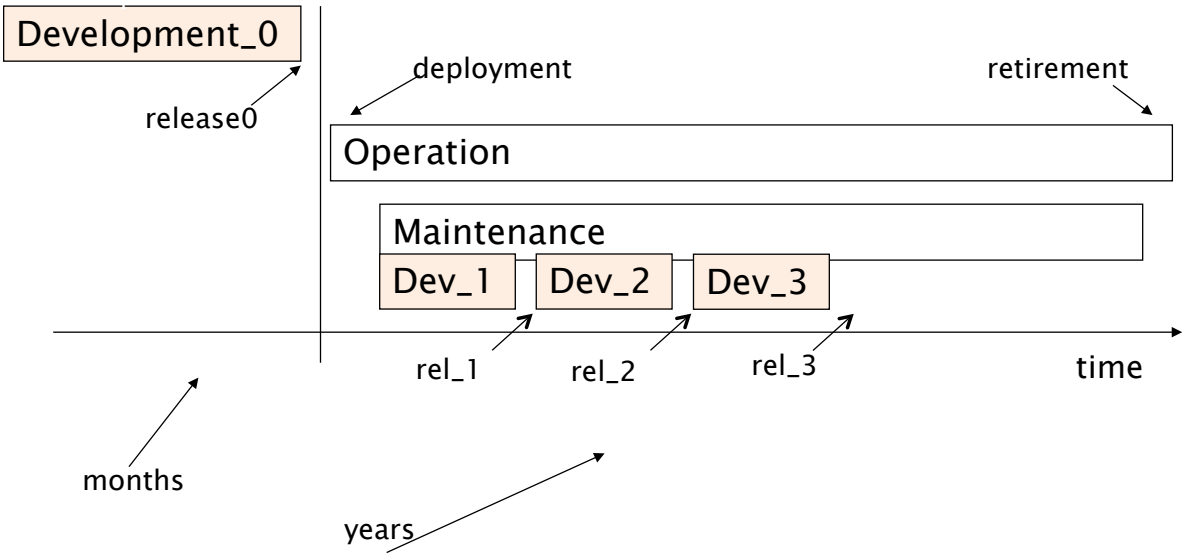
The main phases



Maintenance

- Can be seen as a sequence of developments
- First development usually longer
- Next developments constrained by previous ones and related choices
  - If dev_0 chooses java, next developments are in Java
  - If dev_0 chooses client server model, next developments keep C/S

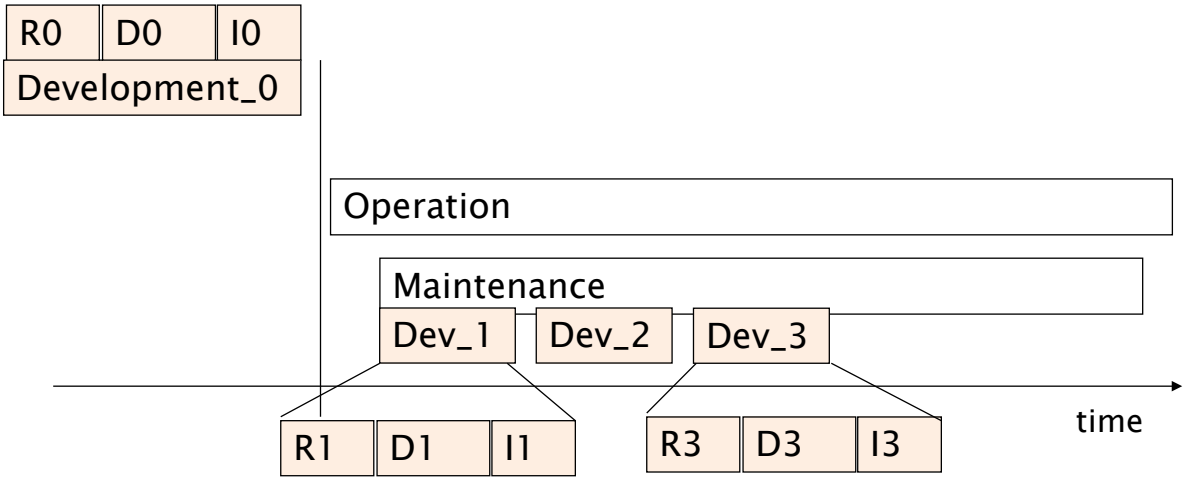
Maintenance



Maintenance

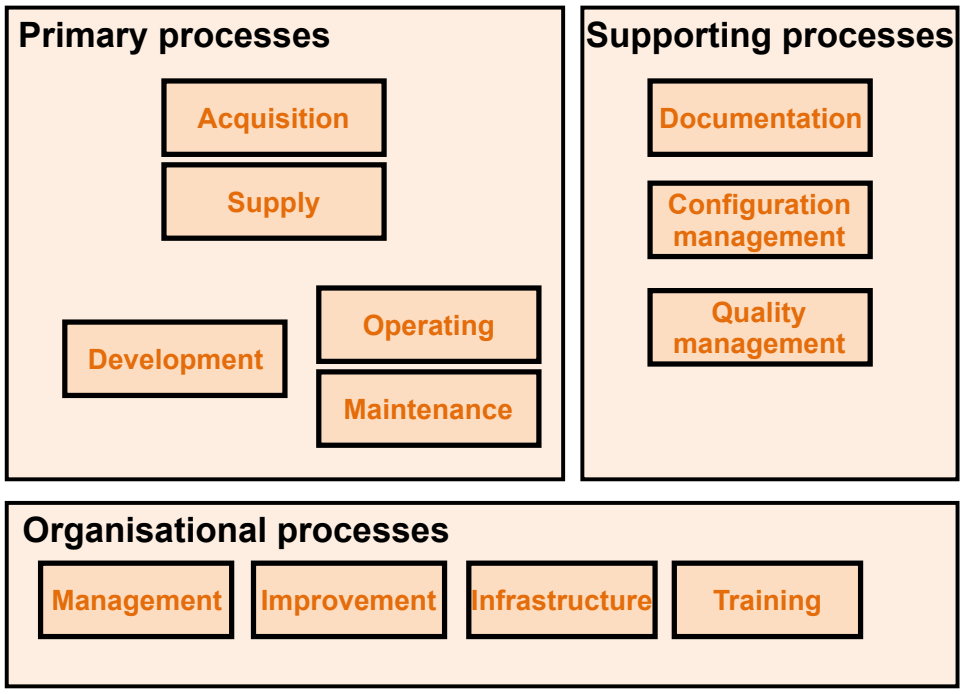
- Development and maintenance do the same activities (requirement, design, etc)
  - But in maintenance an activity is constrained by what has been done before
  - After years, the constraints are so many that changes become impossible

Maintenance



- Development_0
  - Req_0 developed from scratch
  - Design_0 developed from req_0
  - Impl_0 developed from design_0
- Development_1
  - Req_1 from Req_0 (and Des_0, Impl_0)
  - Des_1 from Req_1
  - Impl_1 from Des_1

ISO/IEC 12207



Primary processes

- Acquisition (manage suppliers)
- Supply (interaction with customer)
- Development (develop sw)
- Operation (deploy, operate service)
- Maintenance

Supporting

- Documentation of product
- Configuration management
- Quality assurance
  - Verification and Validation
  - Reviews with customer
  - Internal audits
  - Problem analysis and resolution

Organizational

- Project management
- Infrastructure management
  - Facilities, networks, tools
- Process monitoring and improvement
- Training

Ex. Software development

- Activity 5.3 Software development is decomposed in tasks
  - 5.3.1 Process Instantiation
  - 5.3.2 System requirements analysis
  - 5.3.3 System architecture definition
  - 5.3.4 Software requirements analysis
  - 5.3.5 Software architecture definition
  - 5.3.6 Software detailed design
  - 5.3.7 Coding and unit testing
  - 5.3.8 Integration of software units
  - 5.3.9 Software validation
  - 5.3.10 System integration
  - 5.3.11 System validation

V&V Tasks

- Coding and verification of components (5.3.7.)
- Integration of components (5.3.8.)
- Validation of software (5.3.9.)
- System Integration (5.3.10.)
- System validation (5.3.11.)

Subtasks

- Coding and verification of components (5.3.7.)
  - Definition of test data and test procedures (5.3.7.1.)
  - Execute and document tests (5.3.7.2.)
  - Update documents, plan integration tests (5.3.7.4.)
  - Evaluate tests (5.3.7.5.)
- Integration of components (5.3.8.)
  - Definition of integration test plan (5.3.8.1.)
  - Execute and document tests (5.3.8.2.)
  - Update documents, plan validation tests(5.3.8.4.)
  - Evaluate tests (5.3.8.5.)

ISO 12207

- Only list of activities
- Indipendent of lifecycle
  - Waterfall, iterative, ..
- Indipendent of technology
- Indipendent of application domain
- Indipendent of documentation

How to organize everything?

- Processes
  - Set of related activities
  - To transform input in output
  - Using resources (staff, tools, hw)
  - Within given constraints (norms, standards)

Scenarios in development

- Scenario 1: IT to support businesses
  - Development: several months
  - Operation: years
  - Maintenance: years, up to 60% of overall costs
- Scenario 2: consumer software (games)
  - Development: months
  - Operation: months (weeks)
  - Virtually no maintenance

Scenarios in development

- Scenario 3: Operating System
  - Development: years
  - Operation: years
  - Maintenance: years, up to 60% of overall costs
- Scenario 31: Commercial OS (MS)
  - 2, 3 years to develop
  - Several years maintenance
    - Patches issued every day
    - Major releases (Service Pack) at long intervals
  - In parallel development of a new release
    - Cfr W3.1, 95, NT, 2000, XP, Vista, 7, …

PROCESS MODELS

Three main approaches

- Cow boy programming / Build and Fix
  - Just code, all the rest is time lost and real programmers don't do it
- Document based, semiformal, UML
  - Semiformal language for documents (UML), hand (human) based transformations and controls
- Formal/model based
  - Formal languages for documents, automatic transformations and controls
- Agile
  - Limited use of documents

Models

- Document based
  - Waterfall
  - V
  - Incremental, Evolutionary, Iterative
  - Prototyping
  - Spiral
  - Open source
  - Unified Process – UP – RUP
  - Synch and stabilize
- Agile
  - Scrum, Extreme Programming, Crystal
- Formal methods
  - Formal methods
  - Formal UML

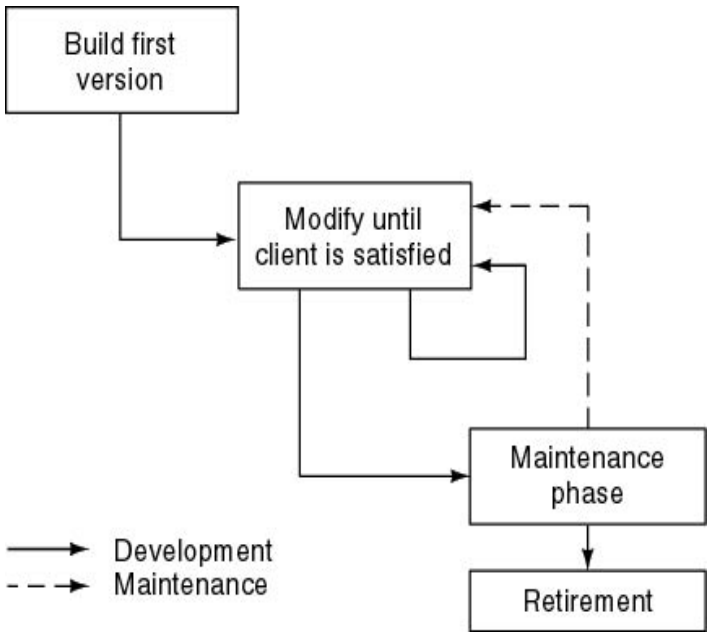
Build and fix

A non-model

- May be ok for solo programming
- Does not scale up for larger projects

- No requirements
- No design
- No validation of requirements/design

Build and fix (code and go)

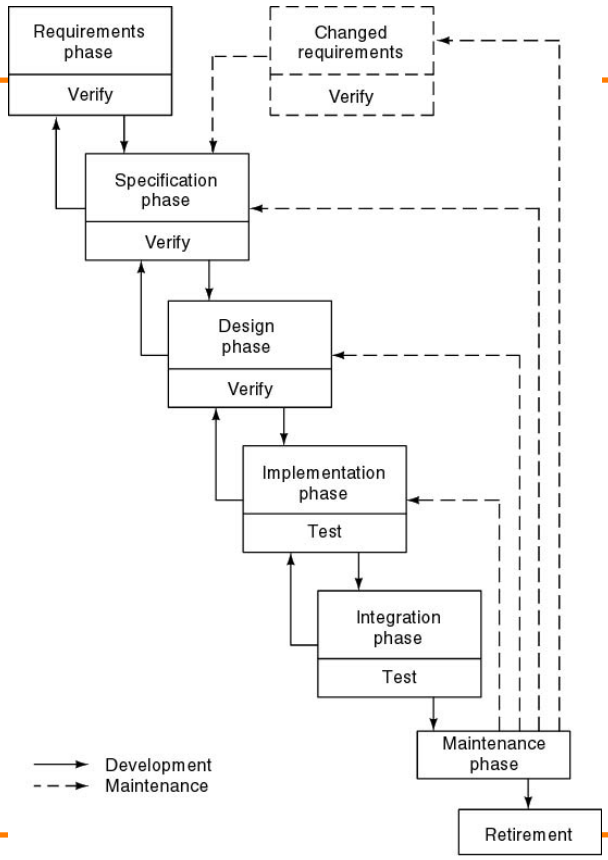


Waterfall

- Sequential activities
  - Activity produces document/deliverable
  - Next activity starts when previous is over and freezes the deliverable
  - Change of documents/deliverables is an exceptional case
- Document driven

[Royce1970]

Waterfall



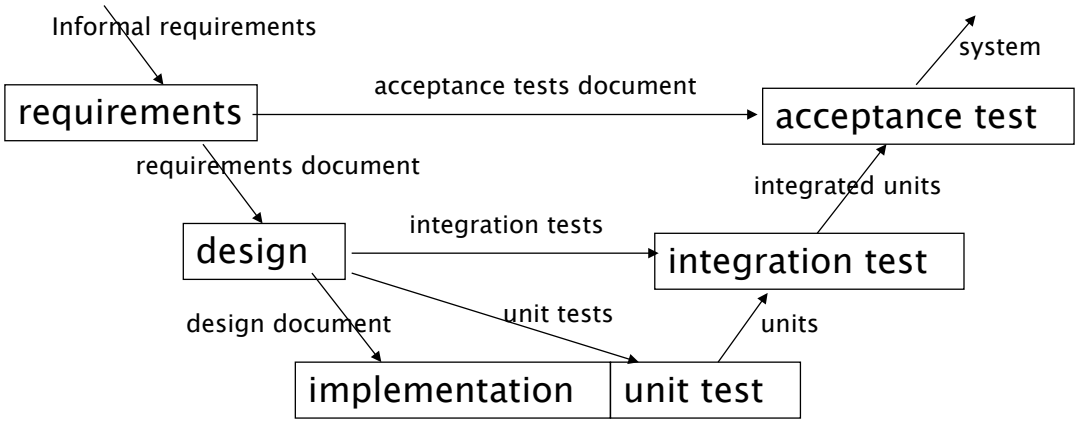
Problems

- Lack of flexibility
  - Rigid sequentiality
  - Requirements supposed to be frozen for long period
    - No changes to improve them
      - Rarely cristal clear
    - No changes to follow changes in context/ customer needs
- Burocratic

V Model

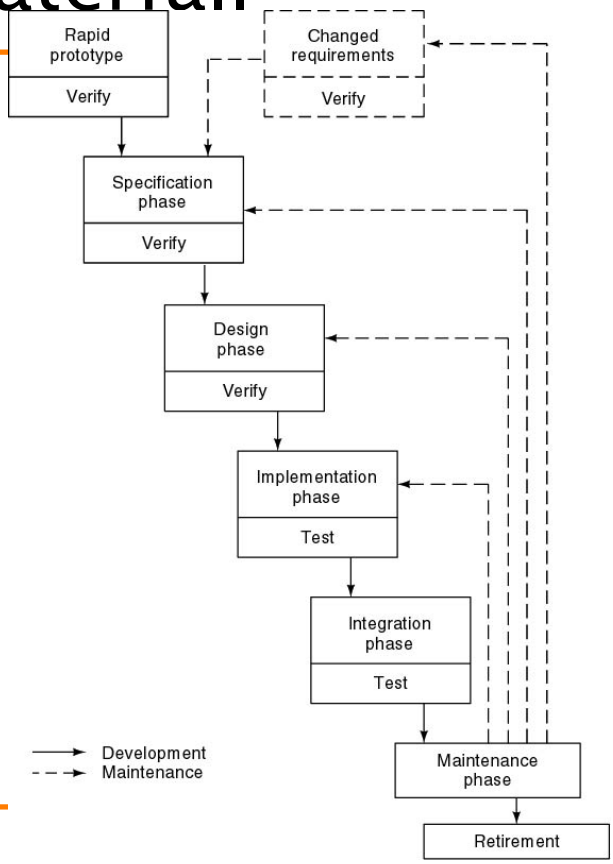
- Similar to waterfall
- Emphasis on V&V activities
- Acceptance tests written after/with requirements
- Unit/integration tests written after/during design

V Model



Prototyping + waterfall

- Quick and dirty prototype to validate/analyze requirements
- Then same as waterfall



Prototyping

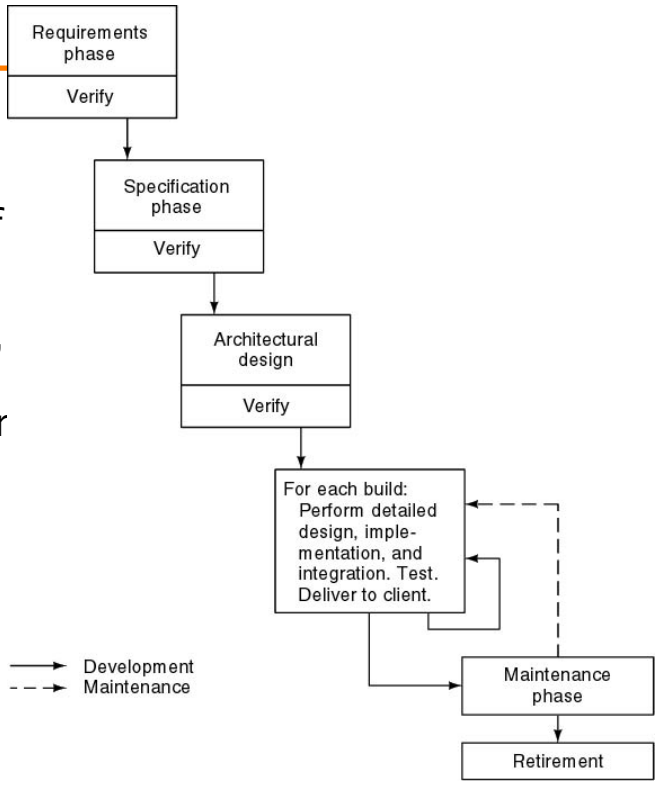
- Advantages
  - Clarify requirements
- Problems
  - Requires specific skills to build prototype (prototyping language)
  - Business pressures to keep prototype (when successful) as final deliverable

Prototype in software

- Subset of functions
- Other language / technology
  - Matlab instead of C
  - Lisp instead of C

Incremental

- Implementation is split increments (builds)
  - Delay implementation of units that depend on external factors (technologies, hardware, etc)
  - Early feedback from user
- Iterations/builds are planned
- Can be associated to prototyping



Evolutionary

- Similar to incremental
- But requirements can change at each iteration
  - Can be associated to prototyping

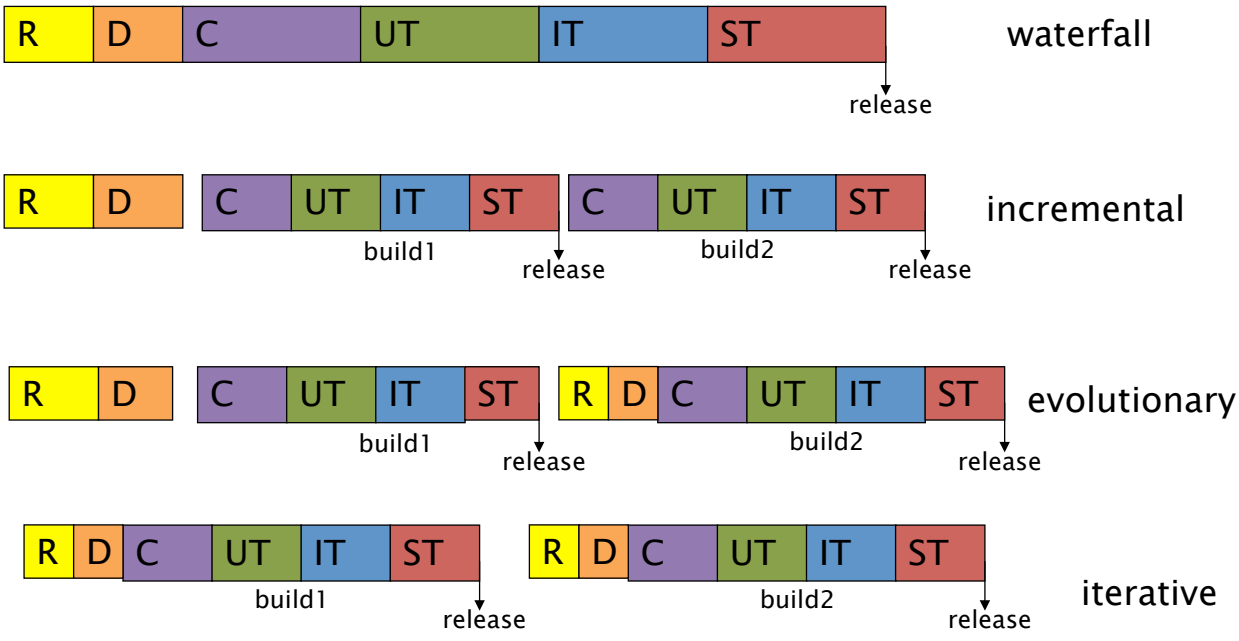
Evolutionary

- Advantages
  - Early feedback, changes to requirements
- Problems
  - Process can become uncontrolled
  - Design may require changes
  - Contractual issues
    - Agreement on effort, not on functions

Iterative

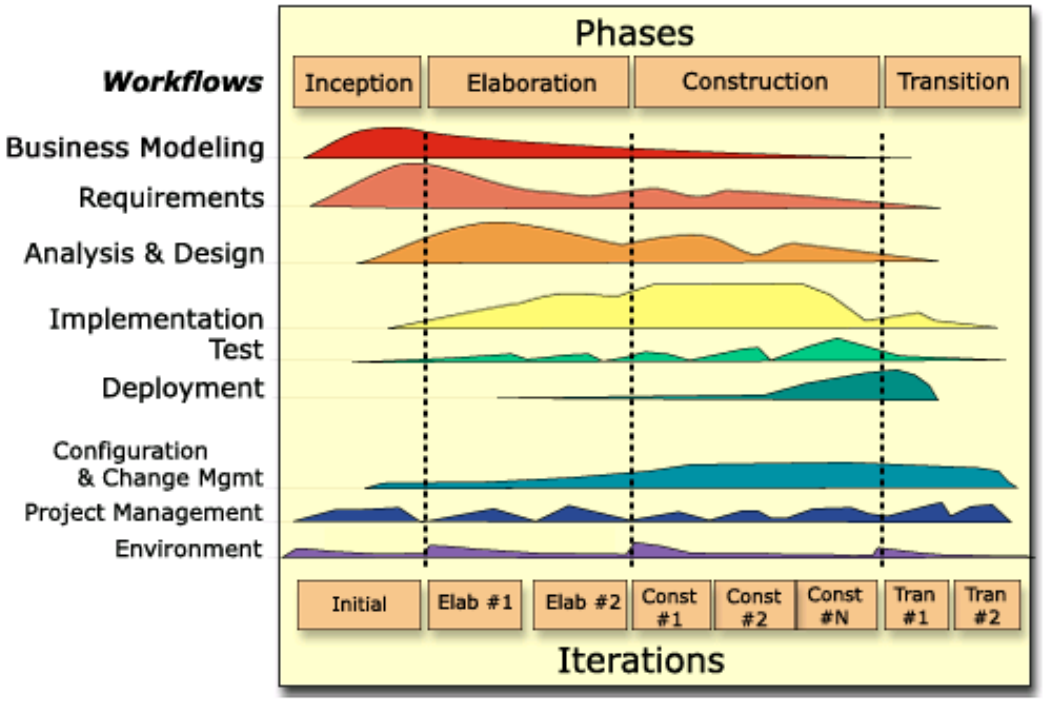
- Many iterations,
- In each iteration a small project (waterfall like)

Processes –comparison



Legend:R= Requirements, D= Design, C=Coding, UT=Unit Test, IT= Integration Test, ST = System Test

(R)UP



(Rational) Unified Process

- Proposed in 1999 by
  - Grady Booch
  - Ivar Jacobson
  - James Rumbaugh
- Characteristics
  - Based on architecture
  - Iterative incremental

UP Phases

- Inception
  - Feasibility study; risk analysis; essential requirements; prototyping (not mandatory)
- Elaboration
  - Requirement analysis; risk analysis; architecture definition; project plan
- Construction
  - analysis, design, implementation, testing
- Transition
  - Beta testing, performance tuning; documentation; training, user manuals; packaging for shipment

Agile manifesto - Values

- Individuals and interactions
  - over processes and tools
- Working software
  - over comprehensive documentation
- Customer collaboration
  - over contract negotiation
- Responding to change
  - over following a plan

Agile Manifesto - Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agile Manifesto - Principles

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity - the art of maximizing the amount of work not done- is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile methods

- XP
- Cristal
- Scrum

Agile Development Principles

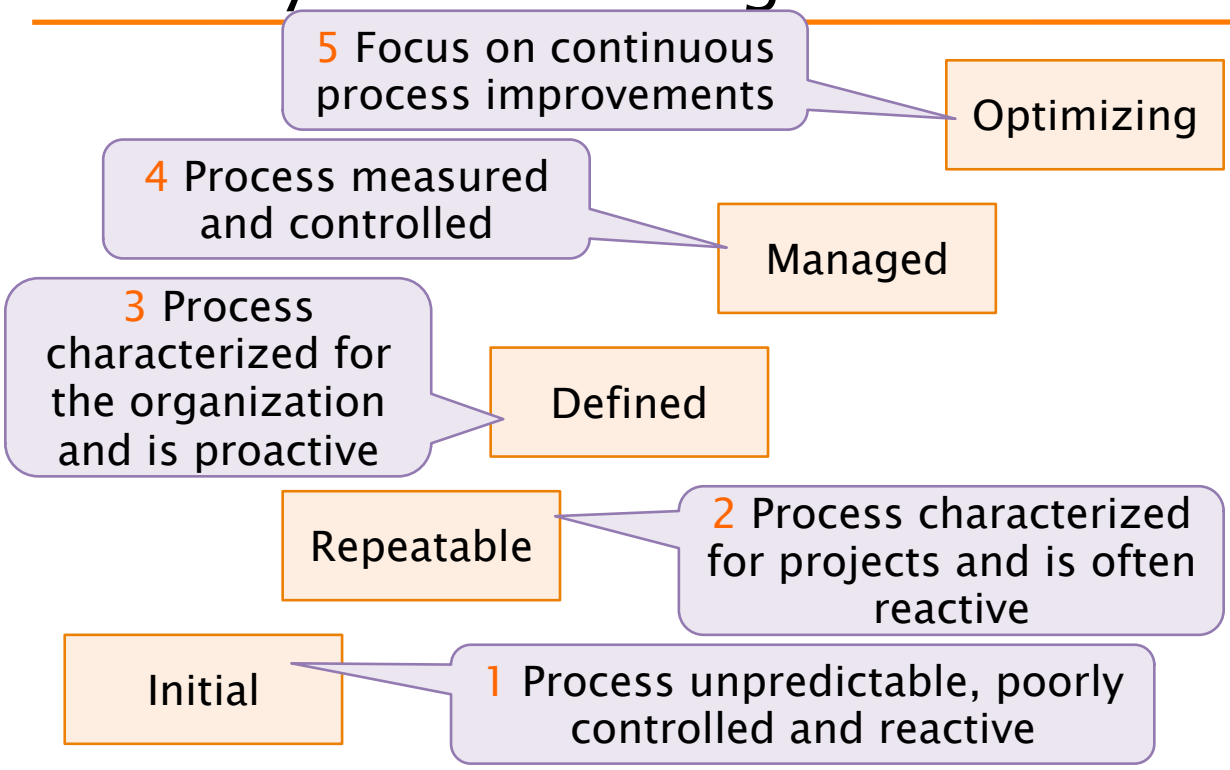
- Test as you go
- Deliver product early and often
  - Feedback
- Document as you go, only as required
- Build cross-functional teams

Assessment/improvement models

- Staged CMMI
- Spice

- Provide a framework to
  - Assess capability
  - Define improvement path in company

Maturity levels for organisation



Levels

1. Initial. The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort.
2. Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
3. Defined. The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.
4. Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.
5. Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Summing up