

Software Engineering

Object Oriented Programming



<https://softeng.polito.it/courses/09CBI>



SoftEng
<http://softeng.polito.it>

Version 2.0.3

© Maurizio Morisio, Marco Torchiano, 2021



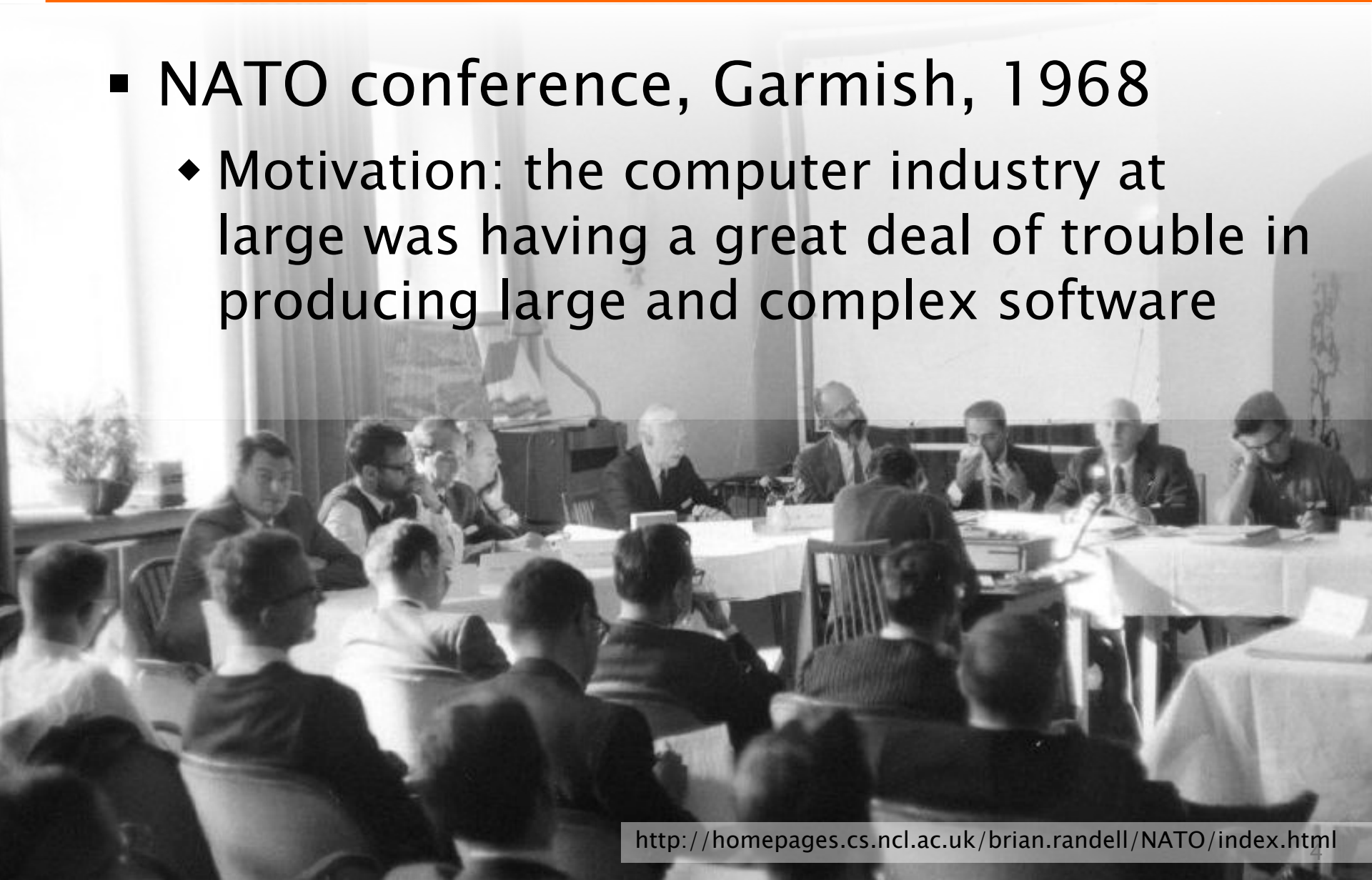
Software Engineering: term

- Margaret Hamilton
 - ◆ standing next to listings of the software she and her MIT team produced for the Apollo project



Software Engineering: discipline

- NATO conference, Garmish, 1968
 - ♦ Motivation: the computer industry at large was having a great deal of trouble in producing large and complex software

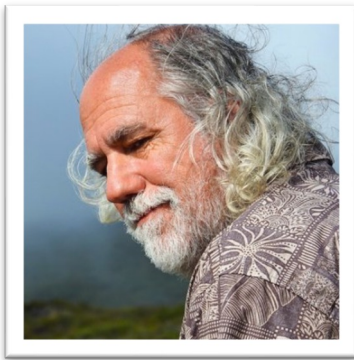


SOFTWARE + ENGINEERING

Software

..is the invisible language that
whispers stories of possibility
to the hardware

– Grady Booch



Engineering

- **Design**: the intentional solution to a problem within a set of constraints
- **Construction**: planning, monitoring, controlling the activities to achieve a solution, tools and techniques
- **Operation**: conduction of the solution and adaptation, within its limits

Software Engineering

Multi person construction of multi version software

- ◆ Dave Parnas



Software vs. Program

Software \neq Program



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

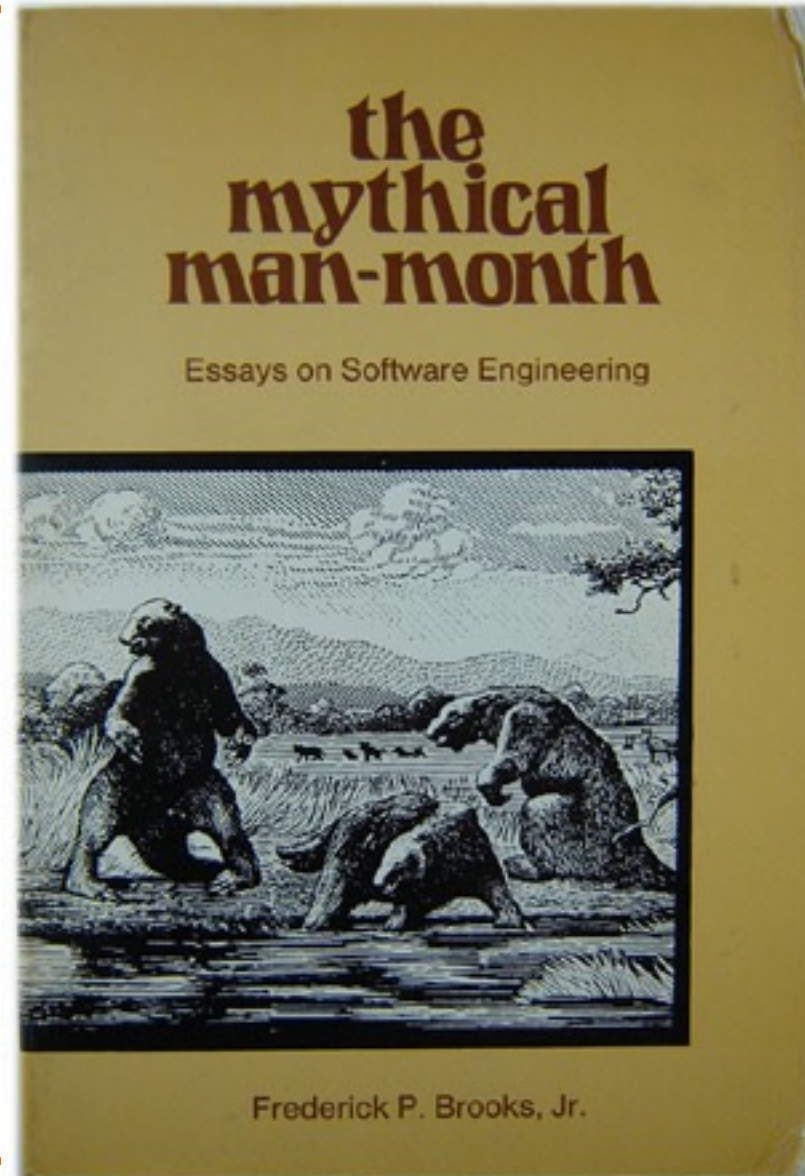
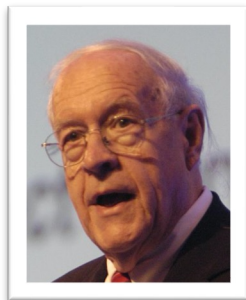
Software Discipline Premises

- Evolutionary and experimental
 - ◆ Software does not age, 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

*Adding manpower to
a late software project
makes it later.*

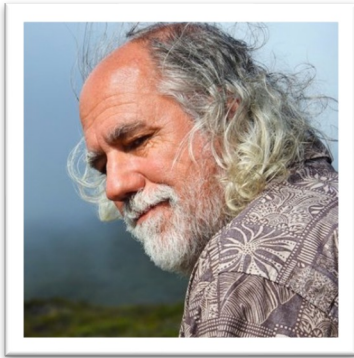
- Fred Brooks, 1975



Ethics

Every line of code has a moral or ethical implication

- ◆ Grady Booch



Ethics

- ACM Code of Ethics and Professional Conduct
 1. General Principles
 2. Professional Responsibilities
 3. Professional Leadership Principles
 4. Compliance with the Code
- ◆ <https://www.acm.org/code-of-ethics>



ACM Code of Ethics and
Professional Conduct

SOFTWARE DEVELOPMENT PROCESS

Goal

Produce software

- ♦ documents, data, code

with defined, predictable

process properties:

- ♦ cost, duration

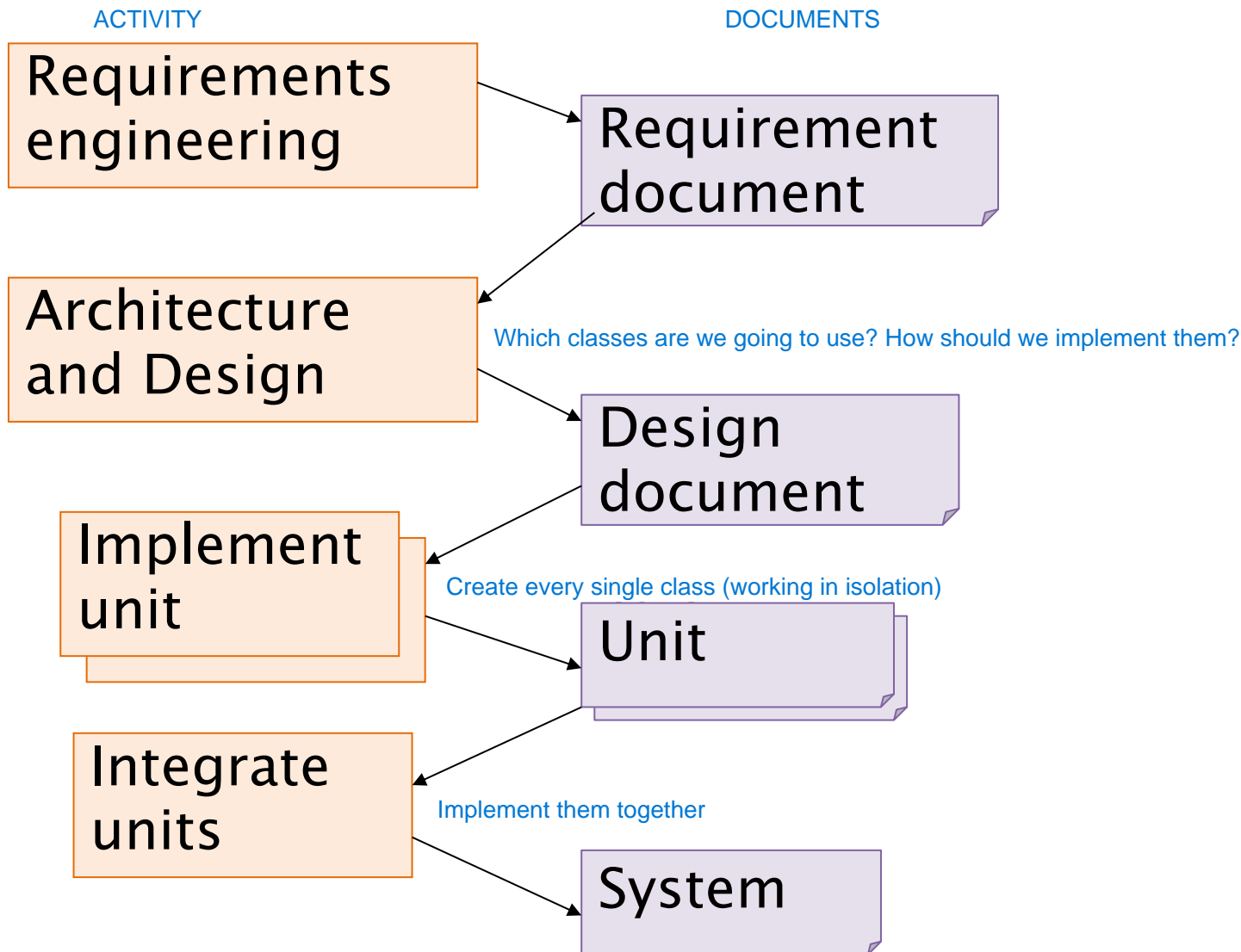
and **product** properties:

- ♦ 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
-

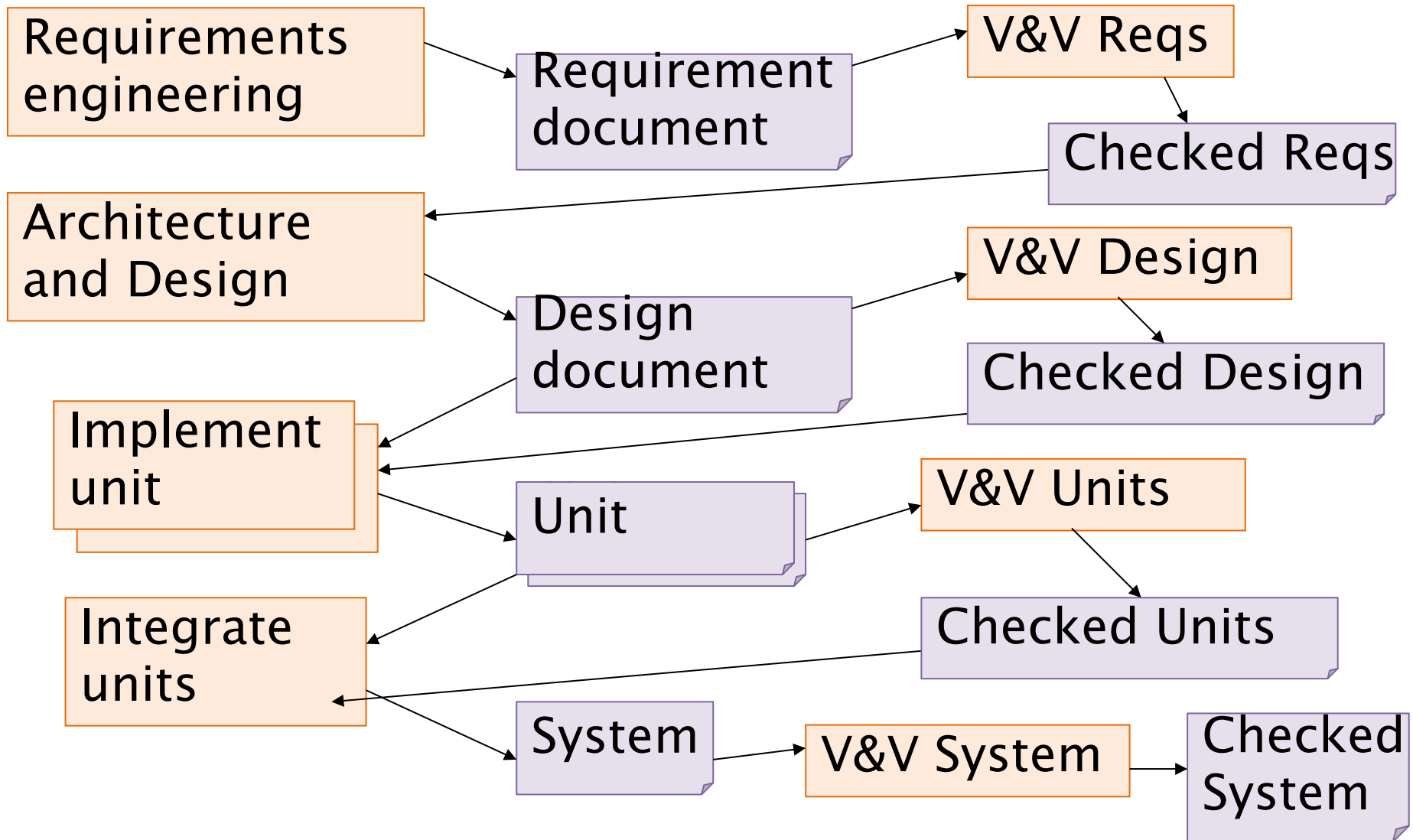
Production activities



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)
-

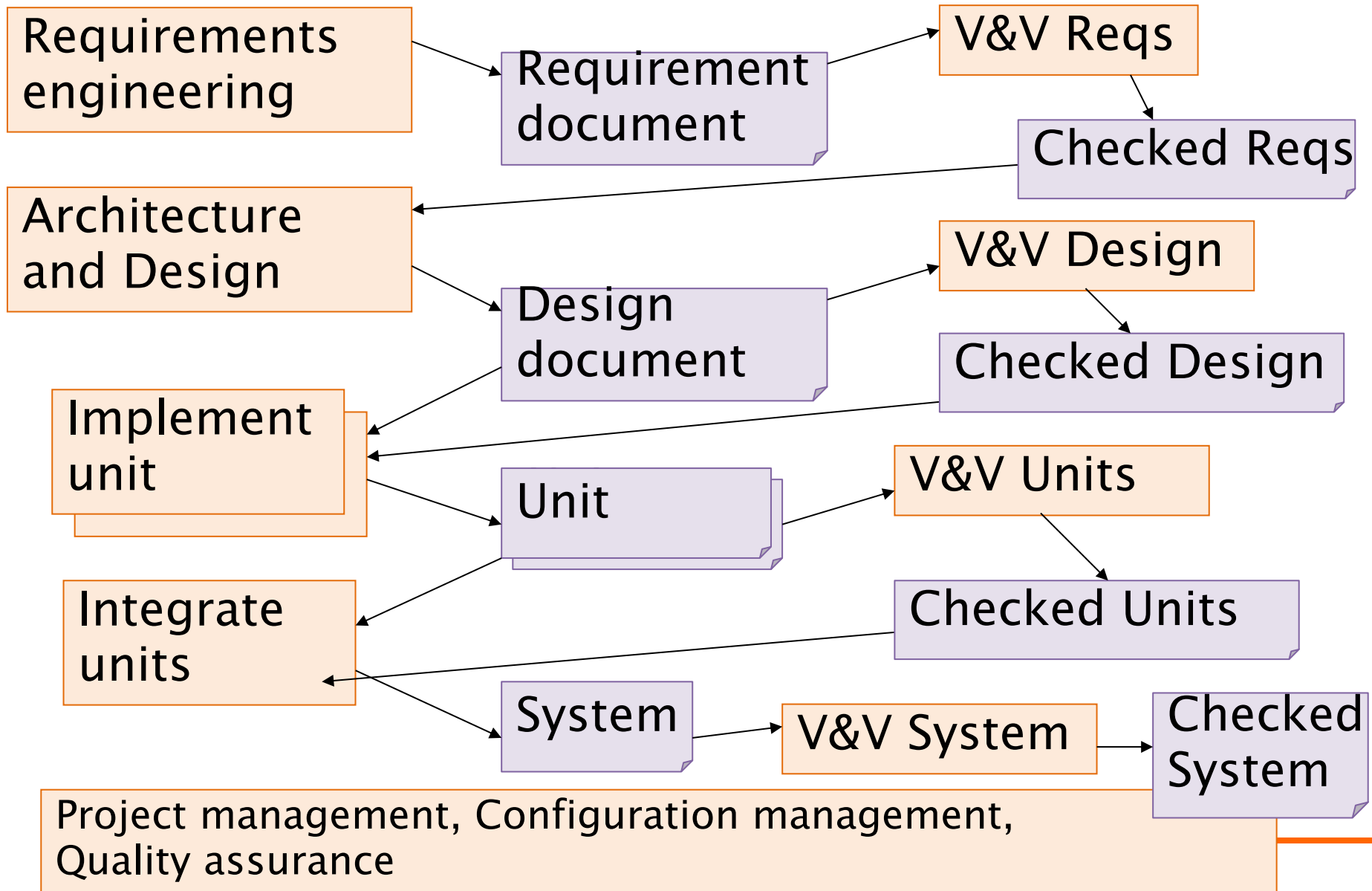
Production activities



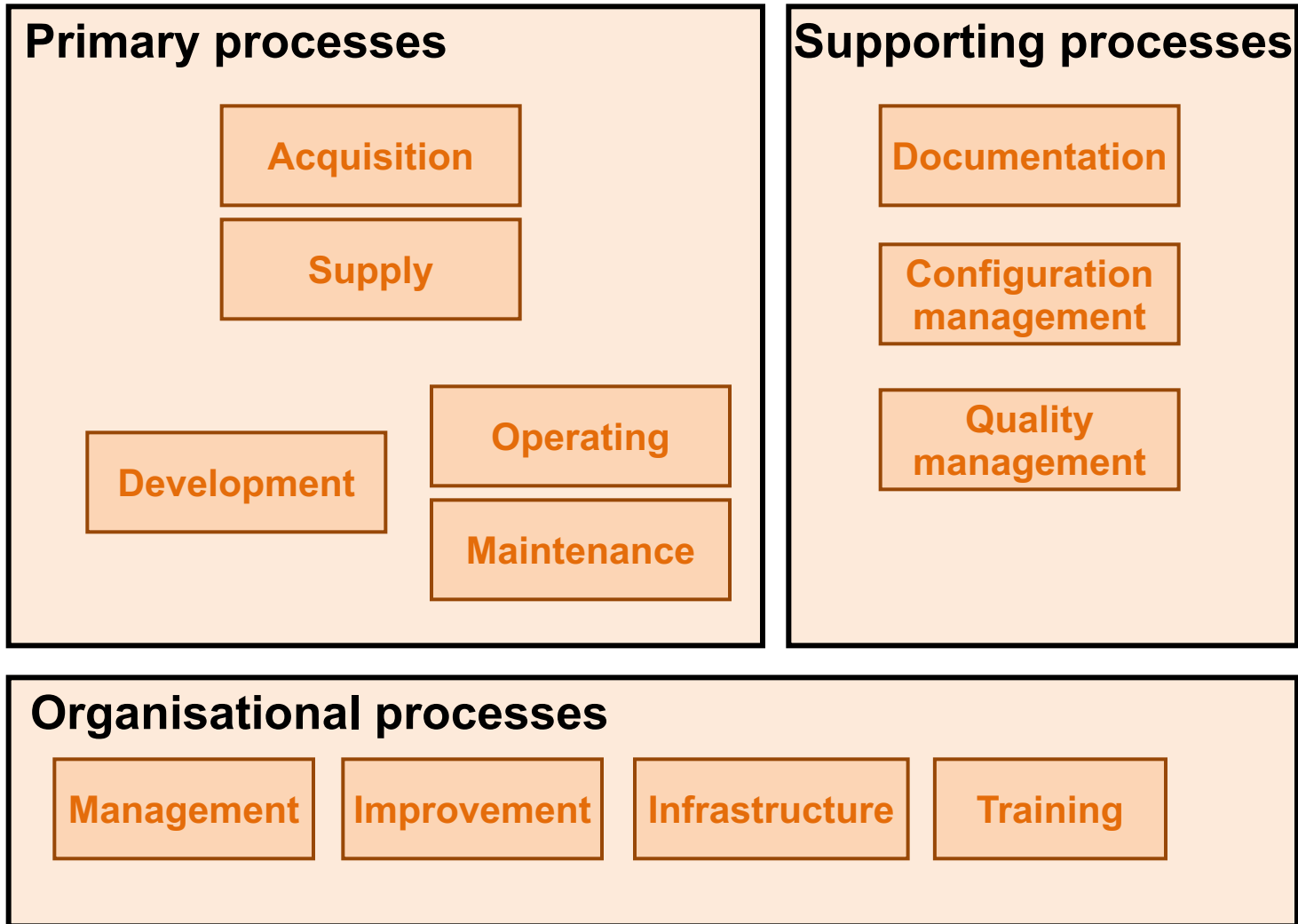
The management activities

- 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



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
-

ISO 12207

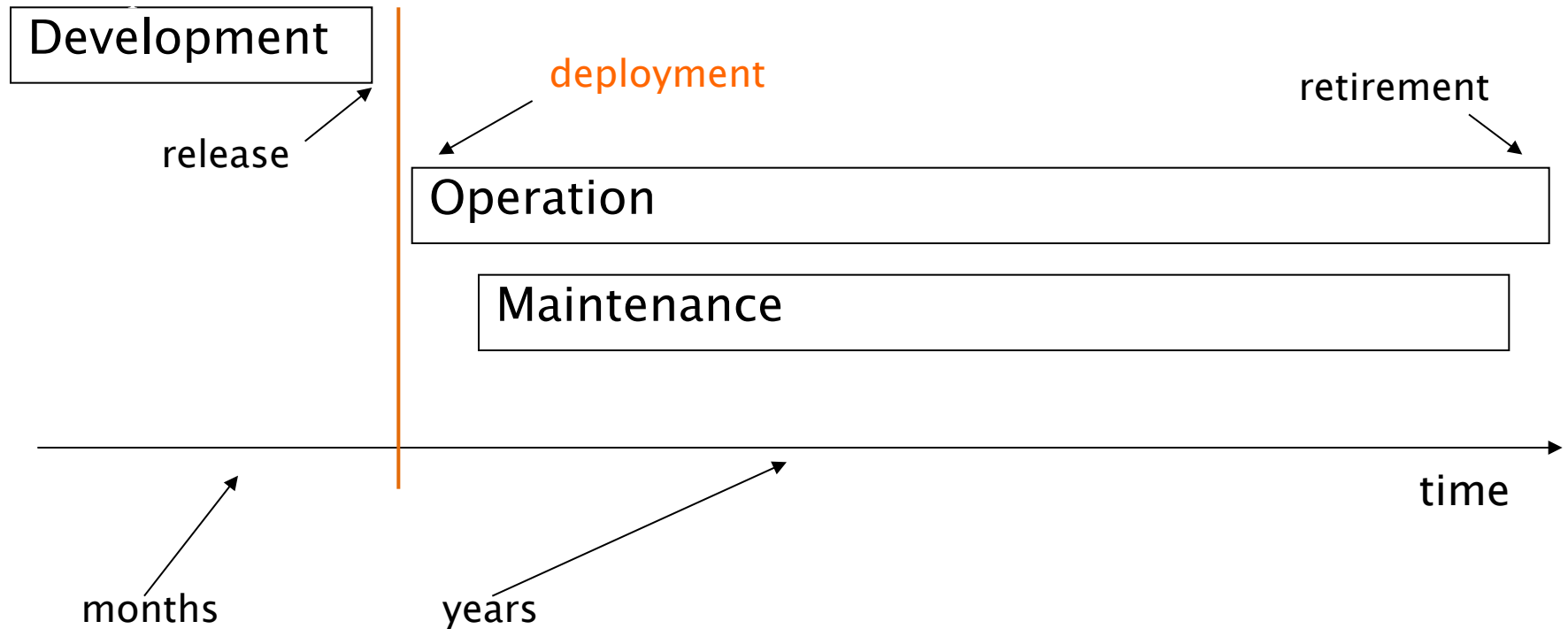
- Only list of activities
 - Independent of lifecycle
 - ♦ Waterfall, iterative, ..
 - Independent of technology
 - Independent of application domain
 - Independent of documentation
-

PHASES

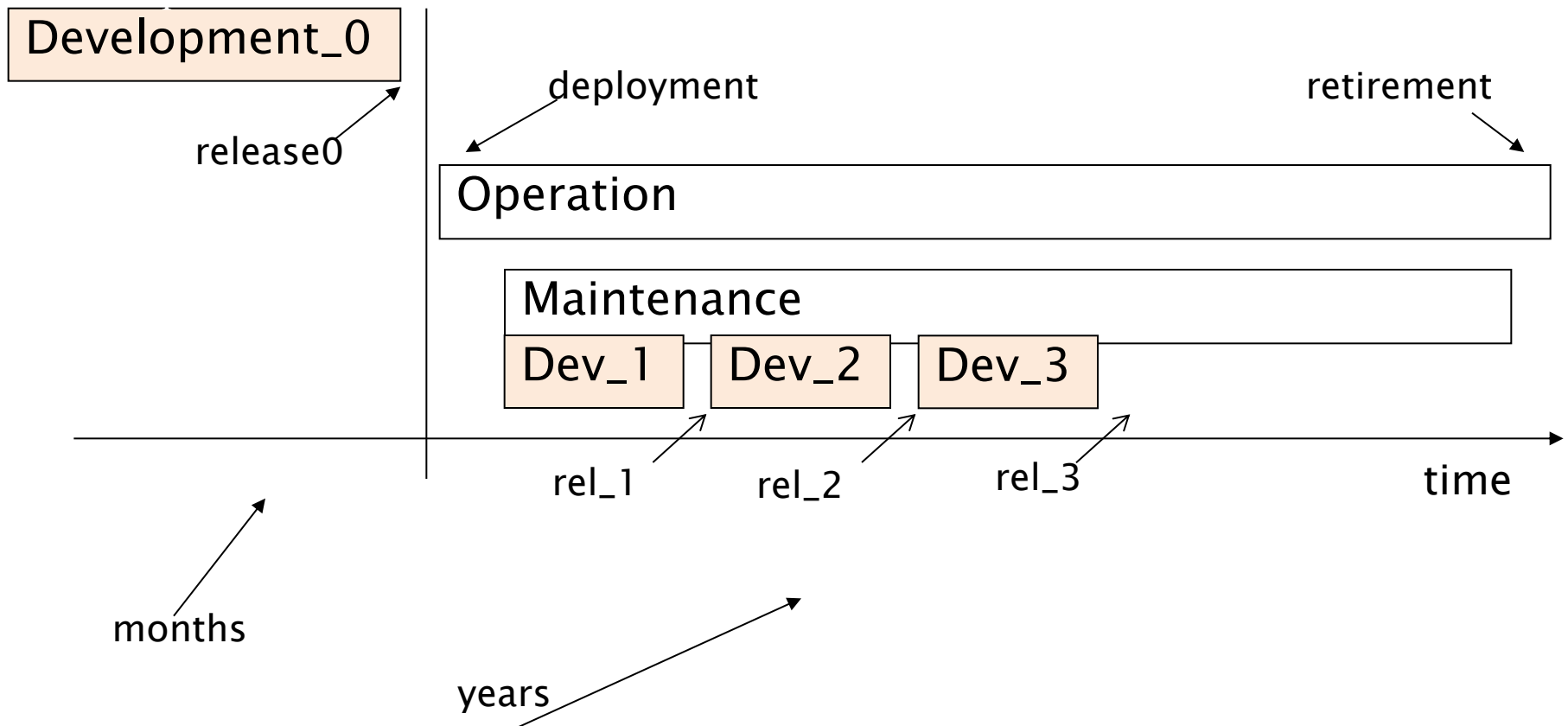
Beyond development

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

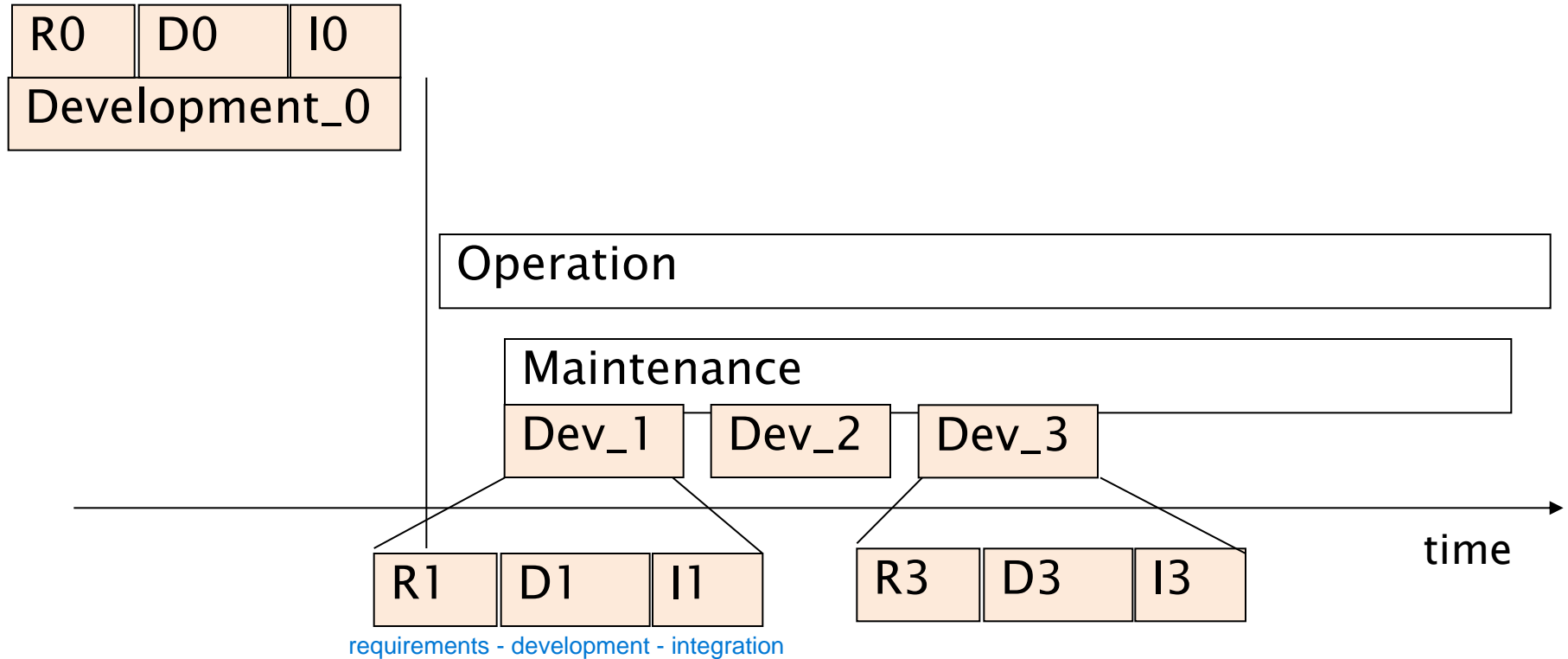
The main phases



Maintenance



Maintenance



PROCESS MODELS

How to organize everything?

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

Three main approaches

- Build and Fix a.k.a. Cow-boy programming
 - ◆ 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

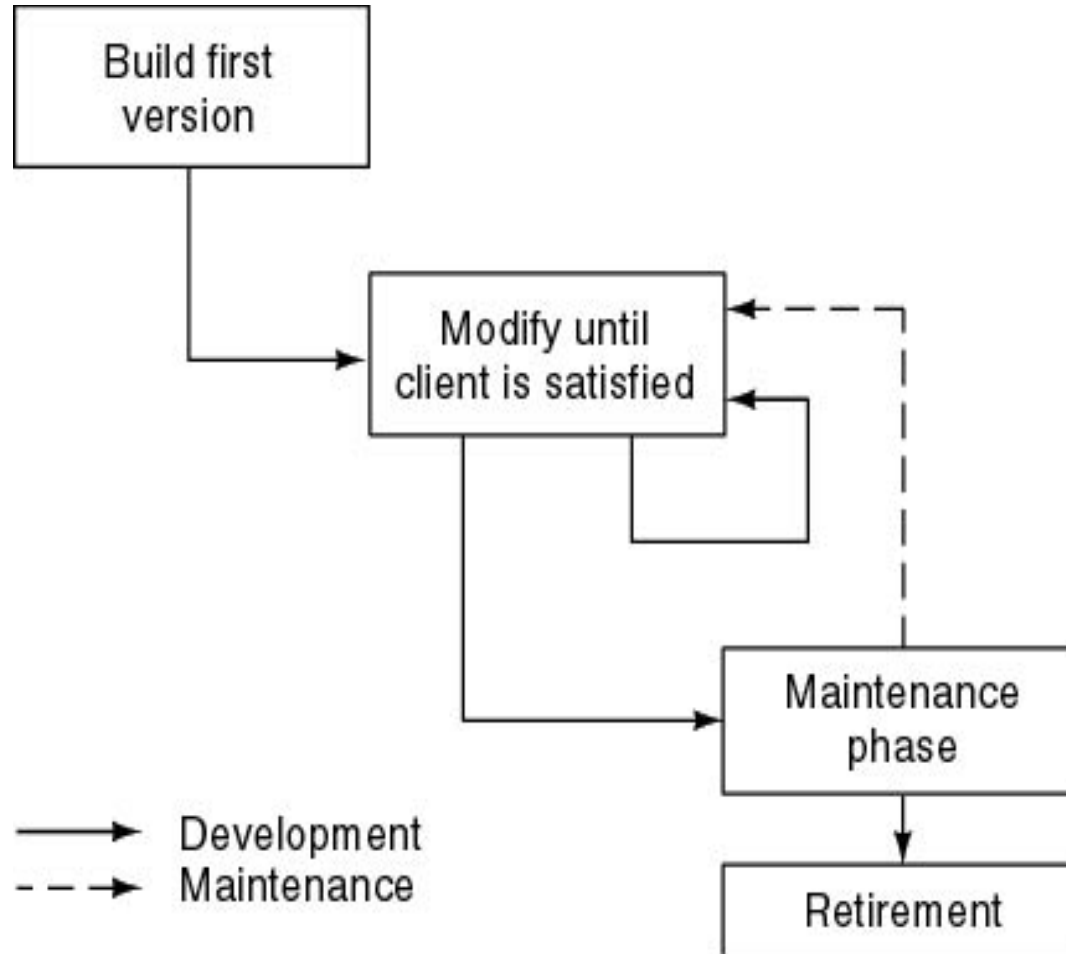


A non-approach

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

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

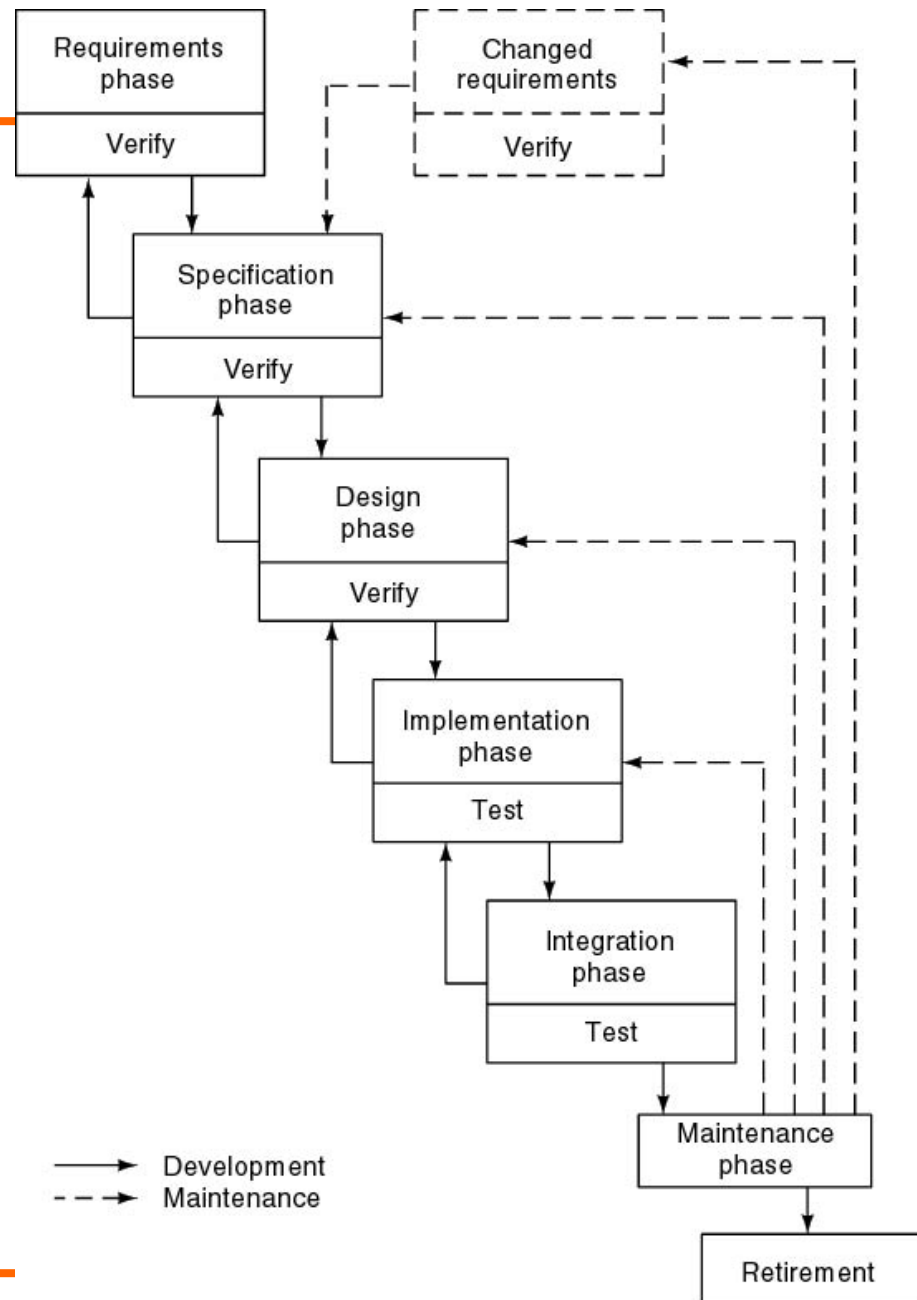
Build and fix



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

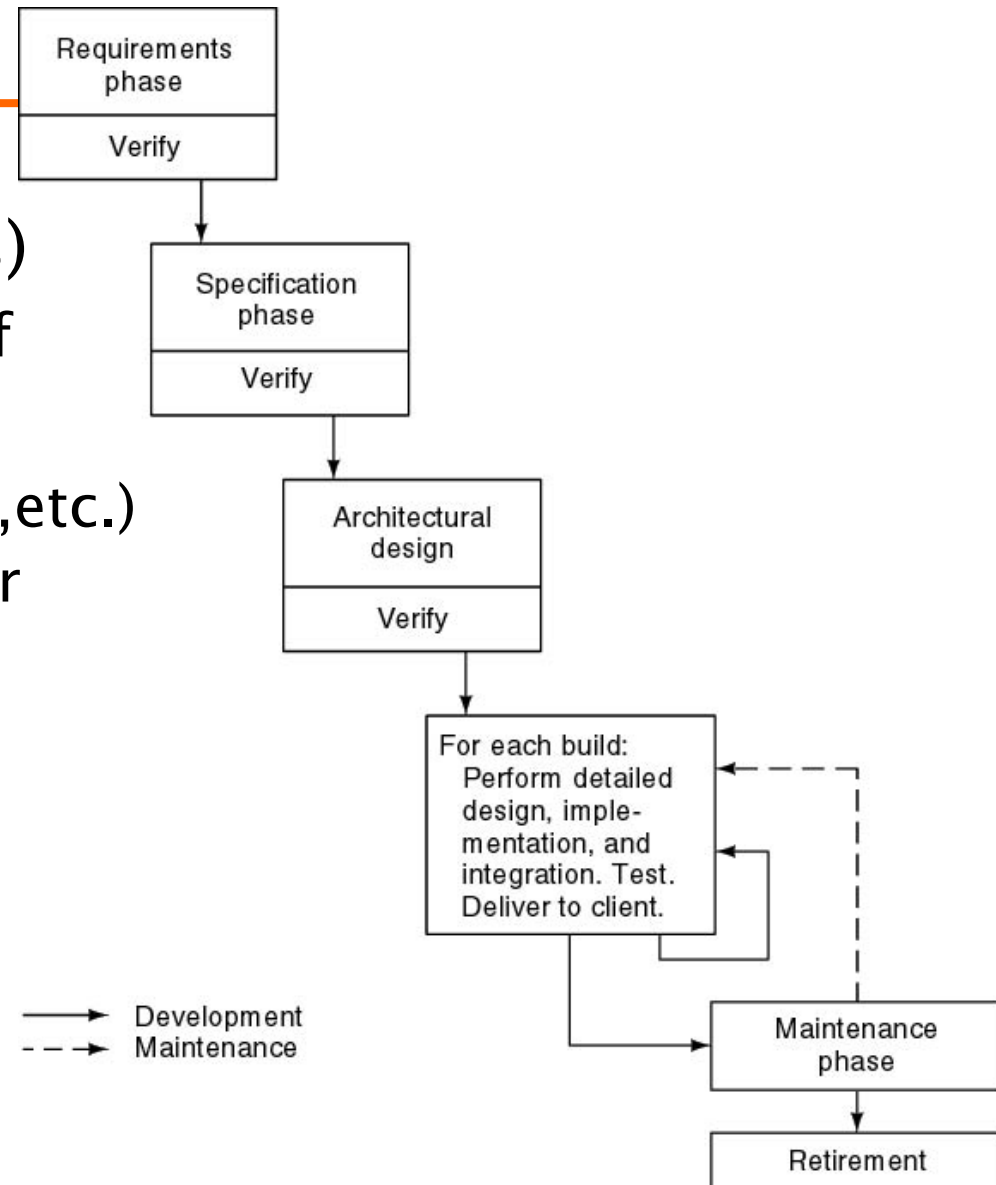


Problems

- 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
-

Incremental

- Implementation is split into 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
-

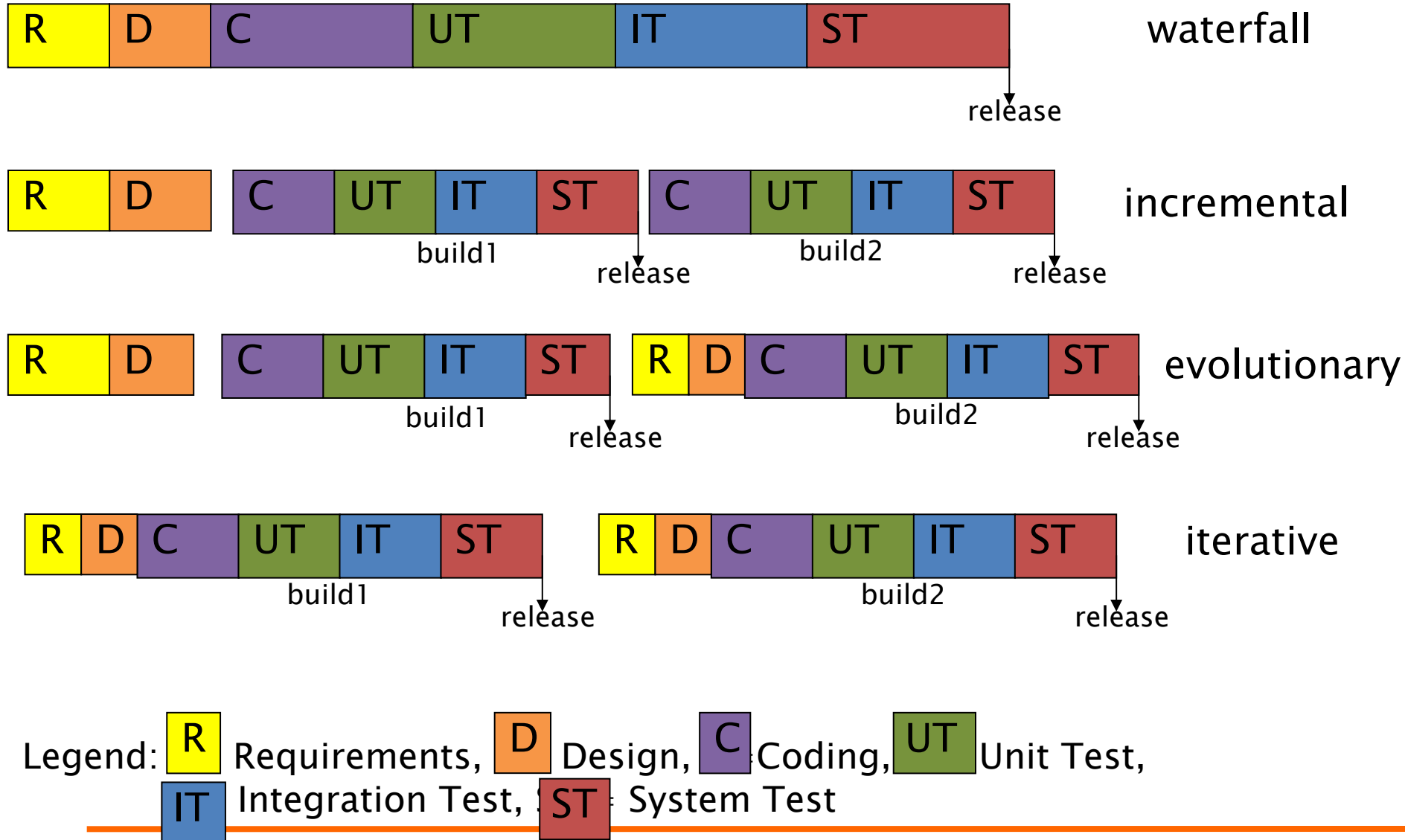
Evolutionary

- 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)

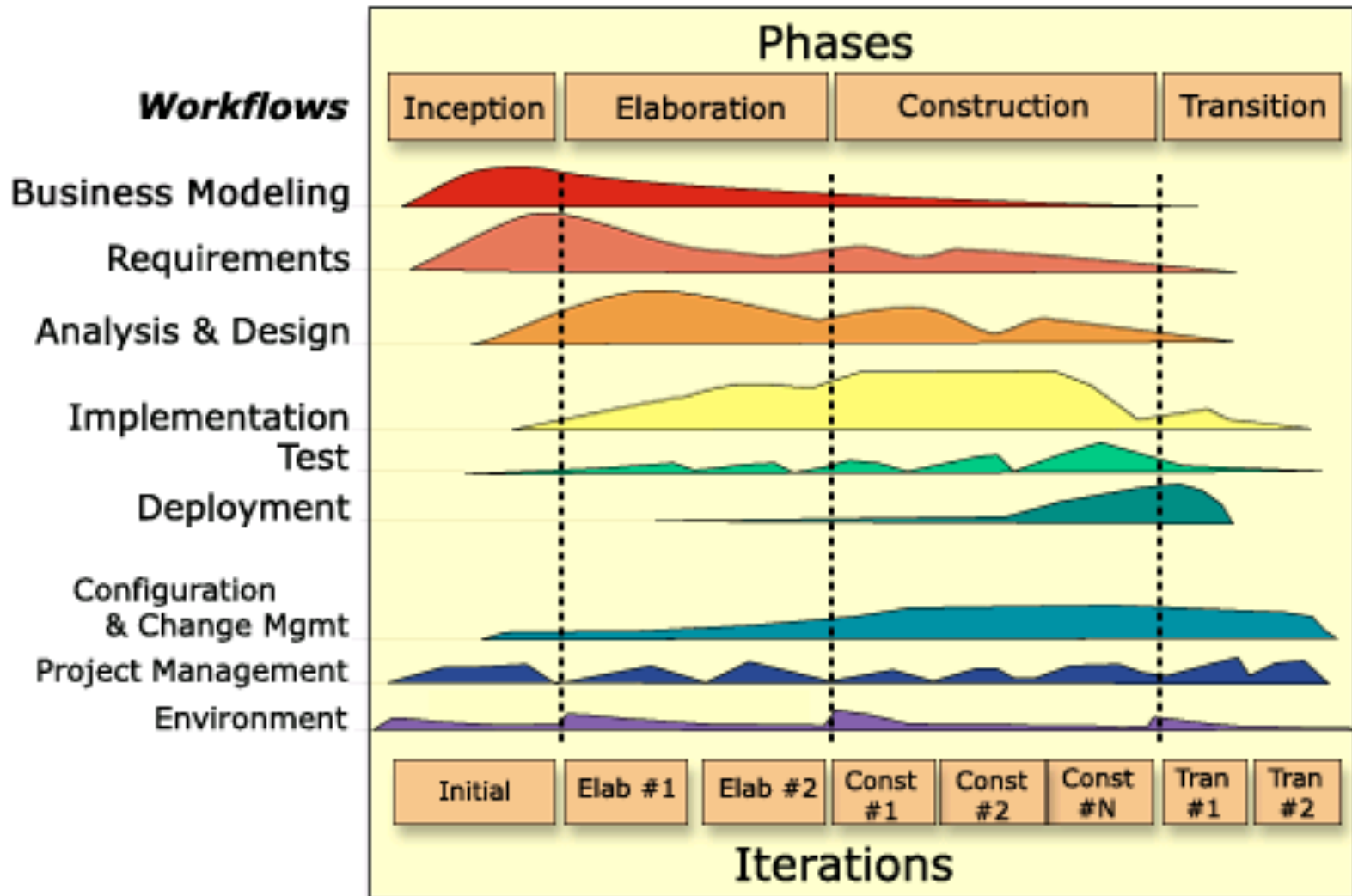
Processes –comparison



(Rational) Unified Process

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

(R)UP



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

5 Focus on continuous process improvements

Optimizing

4 Process measured and controlled

Managed

3 Process characterized for the organization and is proactive

Defined

Repeatable

2 Process characterized for projects and is often reactive

Initial

1 Process unpredictable, poorly controlled and reactive

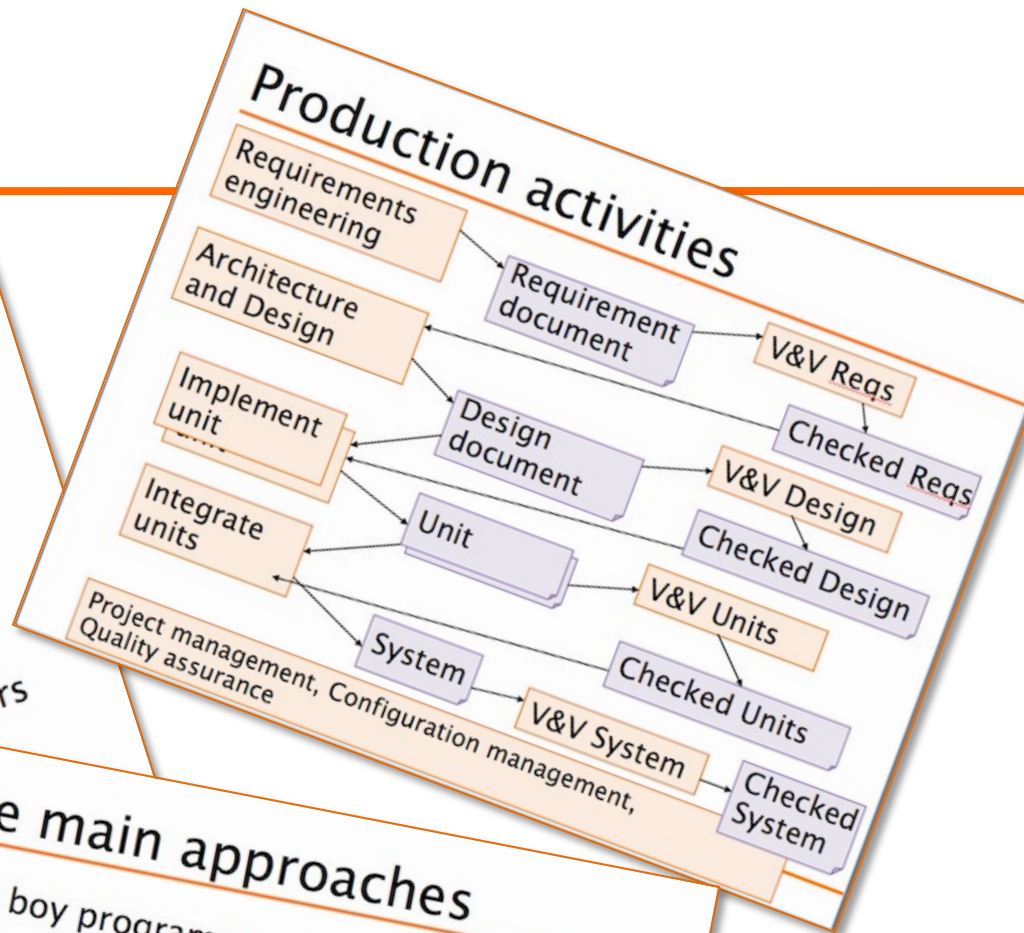
Summing up

Software vs. Program

Software \neq Program

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

SoftEng



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

SoftEng