Politecnico
di Milano

Software Engineering

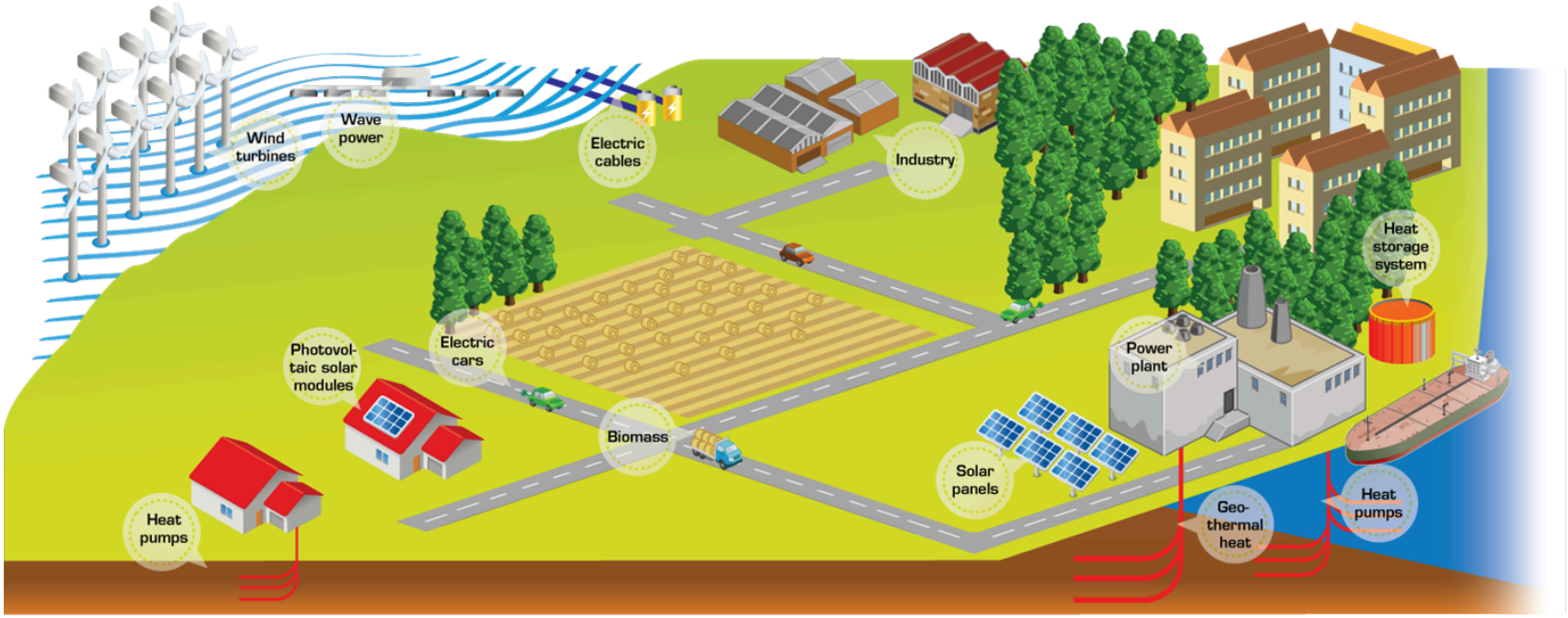
Course introduction
Definitions
History
The process and product
Phases of the development process

Why software engineering is important

- Software is everywhere and our society is now totally dependent on software-intensive systems
- Society could not function without software

Transport systems

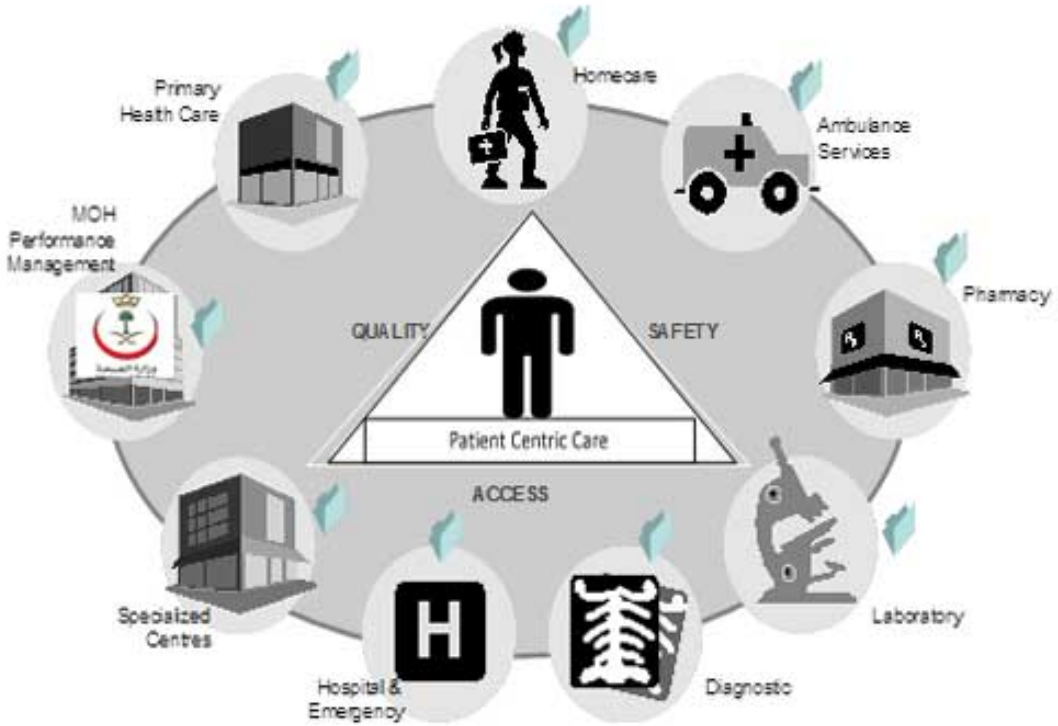
Energy systems



Manufacturing systems

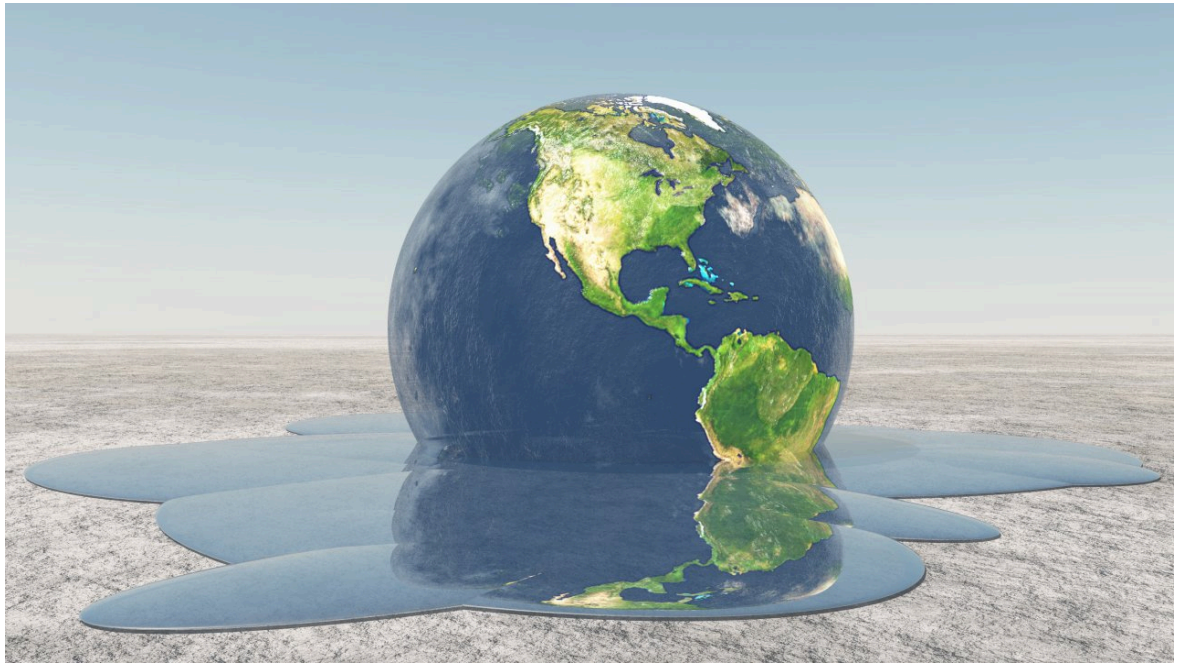
New challenges

- More advanced software is required to address new challenges
  - Increasing number of elderly people



New challenges

- More advanced software is required to address new challenges
  - Increasing number of elderly people
  - Climate change



New challenges

- More advanced software is required to address new challenges
  - Increasing number of elderly people
  - Climate change
  - Combating international terrorism

New challenges

- More advanced software is required to address new challenges
  - Supporting increasing numbers of elderly people
  - Climate change
  - Combating international terrorism

- We need software engineering to help manage the complexity of new intensive software-systems, ensure that these systems are reliable and secure and meet the needs of their users

Software engineering: definitions

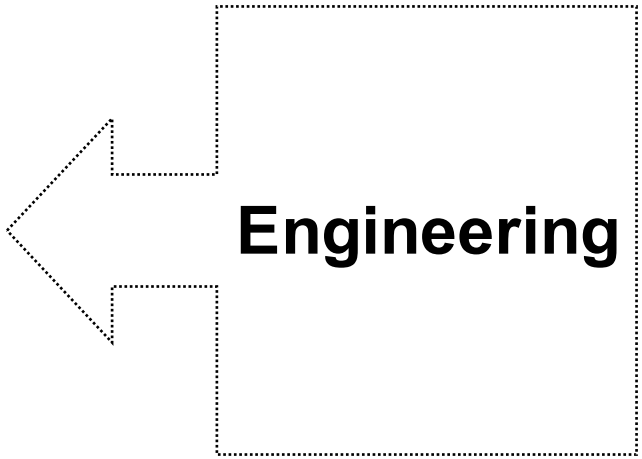
- Field of computer science dealing with software systems
  - large and complex
  - built by teams
  - exist in many versions
  - last many years
  - undergo changes
- Multi-person construction of multi-version software

Software engineering: definitions

- Systematic approach to development, operation, maintenance, deployment, retirement of software
- Methodological and managerial discipline concerning the systematic production and maintenance of software products that are developed and maintained within anticipated and controlled time and cost limits

Software engineering: definitions

- Deals with cost-effective solutions to practical problems by applying scientific knowledge in building software artifacts in the service of the human race
  - *cost-effective*
  - *practical problems*
  - *scientific knowledge*
  - *building things*
  - *service of the human race*



Software engineer: required skills

- **Programming skills not enough**
  - programmer
    - develops a complete program
    - works on known specifications
    - works individually
  - software engineer
    - identifies requirements and develops specifications
    - designs a component to be combined with other components, developed, maintained, used by others; component can become part of several systems
    - works in a team

Skills of software engineers

- Technical
- Project management
- Cognitive
- Enterprise organization
- Interaction with different cultures
- Domain knowledge
- The quality of human resources is of primary importance

History: initial situation

- Software is art
- Computers used for “computing”
  - mathematical problems
  - designers = users
  - no extended lifetime
- The art of “programming”
  - low level languages
  - resource constraints (speed and memory)

From art to craft

- From computing to information management
- Requests for new (custom) software explode
  - users ≠ designers
  - EDP centers and software houses
- New high level languages
- First large projects and first fiascos
  - time and budget, human cooperation failures
  - wrong specifications

From craft to industrial development

- Term “software engineering” defined in a NATO conference in Garmisch, Oct. 1968
- Focuses of software engineering
  - Development methods and standards
  - Planning and management
  - Automation
  - Verifiable quality
  - Componentization
  - ….

More on fiascos… Therac-25, 1985-87

- A computerized radiation therapy machine used in Canada (6 installations) e in USA (5 installations) for cancer therapy
- A software defect caused massive overdoses of radiations
- At least 4 persons died and others have been seriously injured
- http://sunnyday.mit.edu/therac-25.html

More on fiascos… Therac-25, 1985-87

- Reasons for the problem
  - The software has been inherited by a previous model (Therac-20) where hardware control modules did not allow quantities of radiations above the threshold to be emitted
  - Hardware control modules have been eliminated in Therac-25
  - The control software has been developed by a single programmer
  - Task synchronization has been developed in an ad-hoc way
  - No test or formal verification, no software documentation


-00:00:17

More on fiascos… Ariane 5, 1996

- The 4th of June 1996, 40 secs after take off, Ariane 5 broke up and exploded
  - The launcher contained a cluster of satellites for a value of 500M$ (of 1996)
  - The total cost for developing the launcher has been of 8000M$
- The explosion has been caused by a software failure
  - "The failure […] was caused by the complete loss of guidance and attitude information […]. This loss of information was due to specification and design errors in the software of the inertial reference system."
  - "The extensive reviews and tests carried out during the Ariane 5 Development Programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system, which could have detected the potential failure."
- http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html

Dimensions of large systems

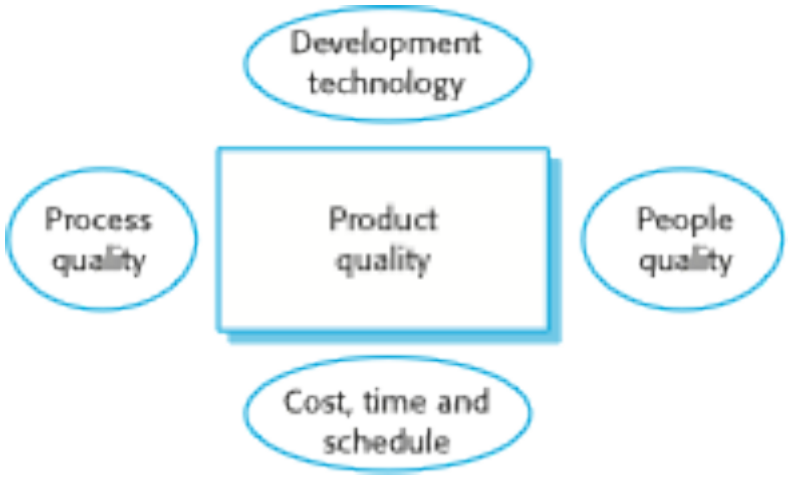
- Examples from Michiel van Genuchten, Towards a Software Factory, Kluwer 1992
  - Space shuttle project
    - 5.6 million code lines, total person year 22k, 1200 Million$
  - CityBank teller machine
    - 780000 code lines, total person year 150, 13.2 Million$
- Debian 5.0 distribution (2007) http://libresoft.dat.escet.urjc.es/debian-counting/
  - 324 million code lines, total person year 122.030, 97, 315000Million$

Process and product

- Our goal is to develop **software products**
- The **process** is how we do it
- Both are extremely important, due to the nature of the software product
- Both have qualities
  - in addition, quality of process affects quality of product
  - …even though, other aspects such as the quality of the development team are important as well

The software product

- Different from traditional types of products
  - intangible
    - difficult to describe and evaluate
  - malleable
  - human intensive
    - does not involve any trivial manufacturing process
- Aspects affecting product quality



Product quality attributes

- Correctness
  - software is correct if it satisfies the specifications
- Reliability
  - can be defined mathematically as “probability of absence of failures for a certain time period”
- Robustness
  - software behaves “reasonably” even in unforeseen circumstances (e.g., incorrect input, hardware failure)
- Performance
  - efficient use of resources
- Usability
  - expected users find the system easy to use

Other qualities

- Maintainability
- Reusability
  - similar to maintainability, but applies to components
- Portability
  - similar to maintainability (adaptation to different target environment)
- Interoperability
  - coexist and cooperate with other applications

Process qualities: productivity

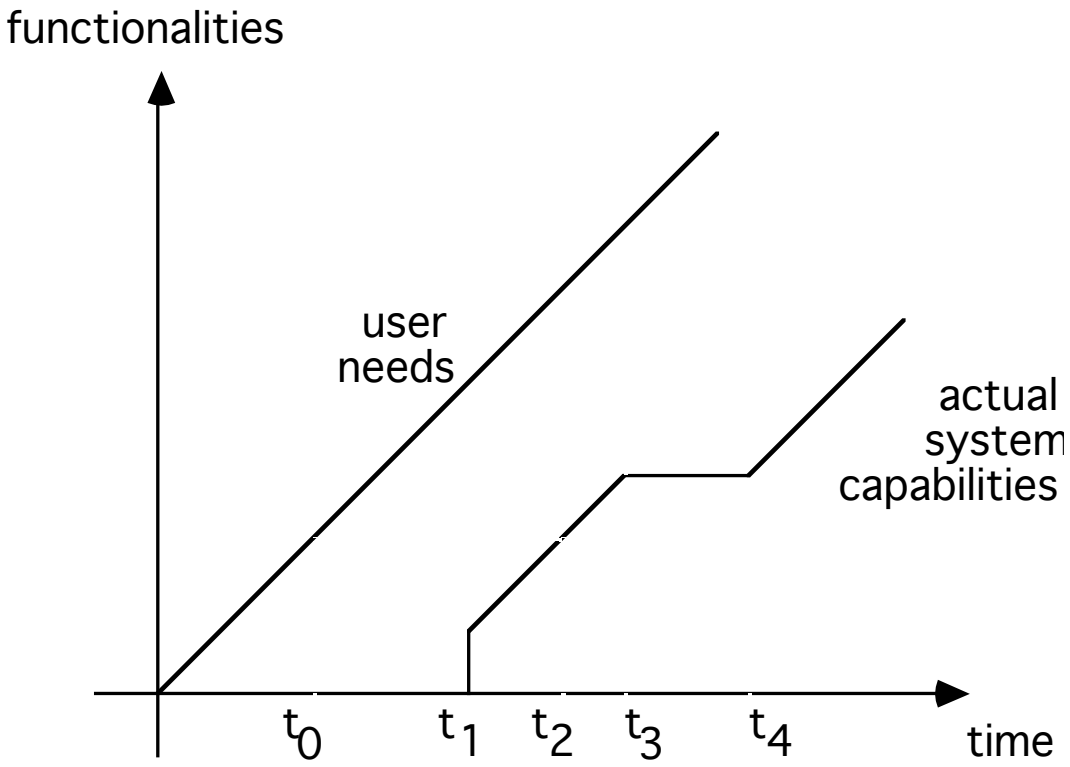
- Productivity
  - how can we measure it?
    - delivered item by a unity of effort
- Unity of effort
  - person month
    - WARNING: persons and months cannot be interchanged
- Delivered item
  - lines of code (and variations)
  - function points

Productivity: folk data

- From the Brooks book "The Mythical Man-Month" (1975)
  - About 1000 lines of code per year per person
- Looking at numbers in slide 23
  - Space shuttle: 255 lines of code per year per person
  - Citybank: 5200 lines of code per year per person
  - Debian : 2600 lines of code per year per person
- Then
  - extreme variance among individuals
  - extreme variance with group dynamics
    - Brooks "law": "adding people to a late project makes the project late(er)"

Process qualities: timeliness

- Ability to respond to change requests in a timely fashion



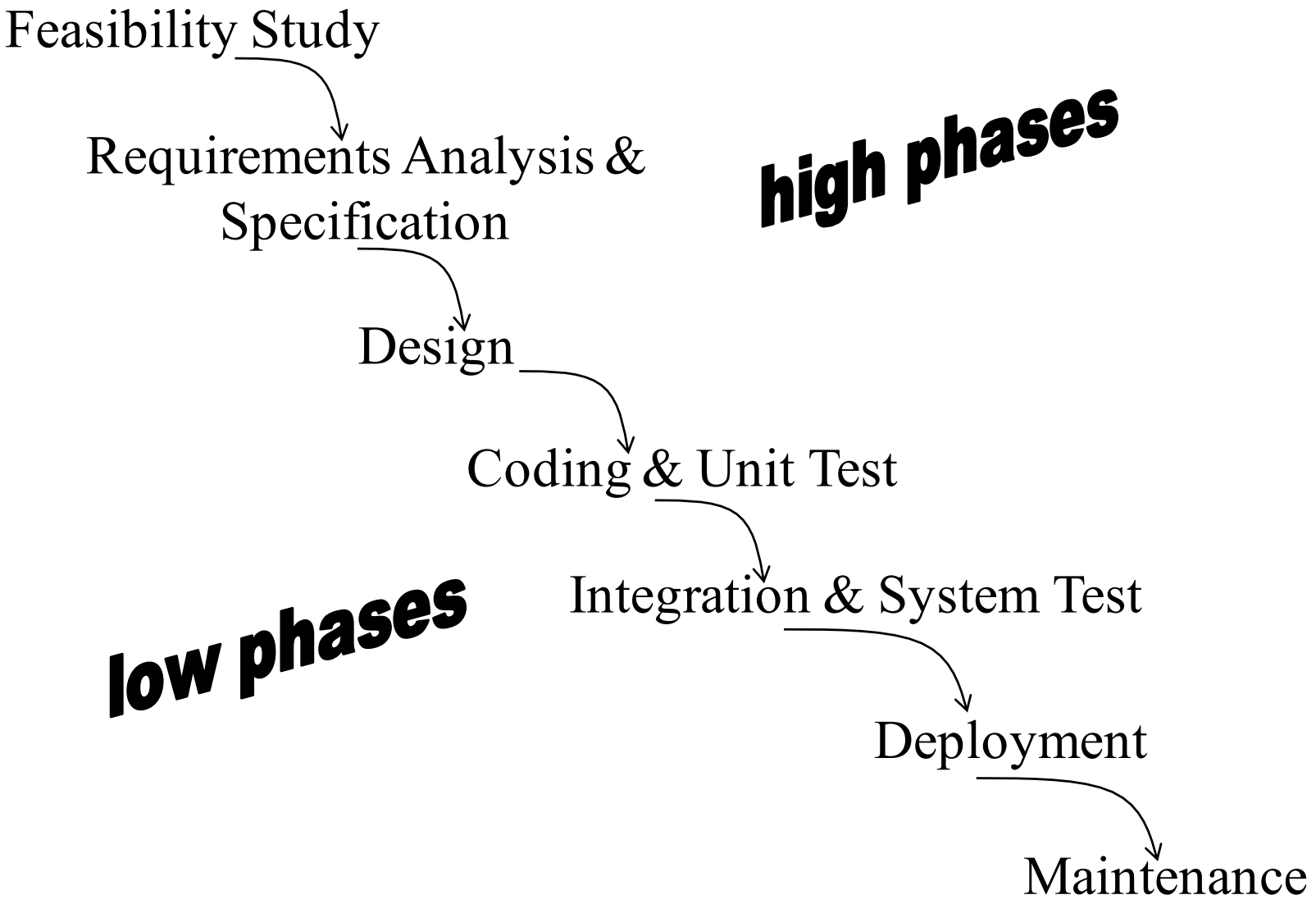
Phases of software development (1)

- In some cases, no reference model:
  - code&fix

Phases of software development (2)

- In some cases, no reference model:
  - code&fix
- The traditional “waterfall” model
  - identify phases and activities
  - force linear progression from a phase to the next
  - no returns (they are harmful)
    - better planning and control
  - standardize outputs (artifacts) from each phase
- Software like manufacturing

A waterfall organization



Feasibility study & project estimation

- Cost/benefits analysis
- Determines whether the project should be started (e.g., buy vs make), possible alternatives, needed resources
- Produces a **Feasibility Study Document**
  - Preliminary problem description
  - Scenarios describing possible solutions
  - Costs and schedule for the different alternatives

Req. analysis and specification

- Analyze the domain in which the application takes place
- Identify requirements
- Derive specifications for the software
  - Requires an interaction with the user
  - Requires an understanding of the properties of the domain
- Produces a **Requirements Analysis and Specification Document (RASD)**

Design

- Defines the software architecture
  - Components (modules)
  - Relations among components
  - Interactions among components
- Goal
  - Support concurrent development, separate responsibilities
- Produces the **Design Document**

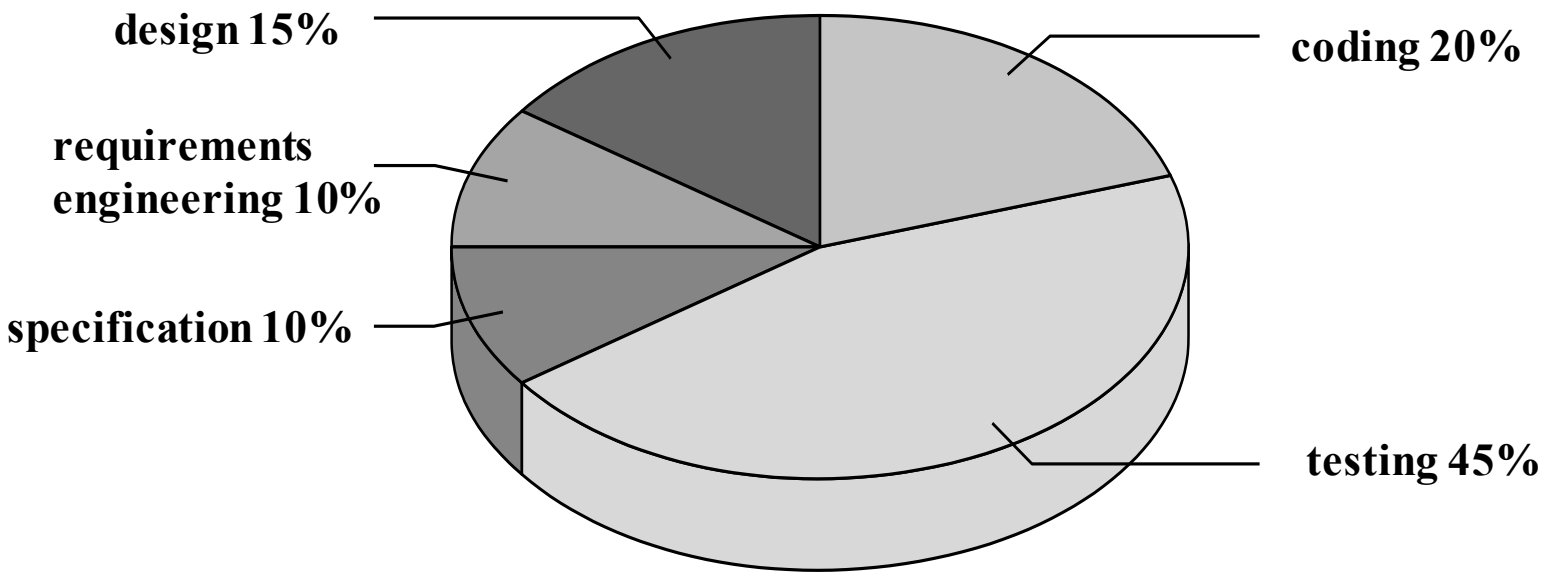
Coding&Unit level quality assurance

- Each module is implemented using the chosen programming language
- Each module is tested in isolation by the module developer
- Inspection can be used as an additional quality assurance approach
- Programs include their documentation

Integration&System test

- Modules are integrated into (sub)systems and integrated (sub)systems are tested
- This phase and the previous may be integrated in an incremental implementation scheme
- Complete system test needed to verify overall properties
- Sometimes we have ***alpha test*** and ***beta test***

Effort distribution (van Vliet 2008)



Software changes

- Good engineering practice
  - first modify design, then change implementation
  - apply changes consistently in all documents
- Software is very easy to change
  - often, under emergency, changes are applied directly to code
  - inconsistent state of project documents
- Software maintenance is (almost) never anticipated and planned; this causes disasters

Maintenance

- Includes different types of change: correction + evolution
  - **corrective** maintenance: deals with the repair of faults or defects found ≈ 20%
  - **adaptive** maintenance: consists of adapting software to changes in the environment (the hardware or the operating system, business rules, government policies…) ≈ 20%
  - **perfective** maintenance: mainly deals with accommodating to new or changed user requirements ≈ 50%
  - **preventive** maintenance: concerns activities aimed at increasing the system's maintainability ≈ 5%

Correction vs evolution

- Distinction can be unclear, because specifications are often incomplete and ambiguous
- This causes problems because specs are often part of a contract between developer and customer
  - early frozen specs can be problematic, because they are more likely to be wrong

Why evolution?

- Wrong specifications (requirements were not captured correctly or domain poorly understood)
- Context changes (adaptive maintenance)
  - EURO vs national currencies
- Requirements change (perfective maintenance)
    - New demands caused by introduction of the system
    - Recent survey among EU companies indicates that 20% of user requirements are obsolete after 1 year
- Requirements not known in advance

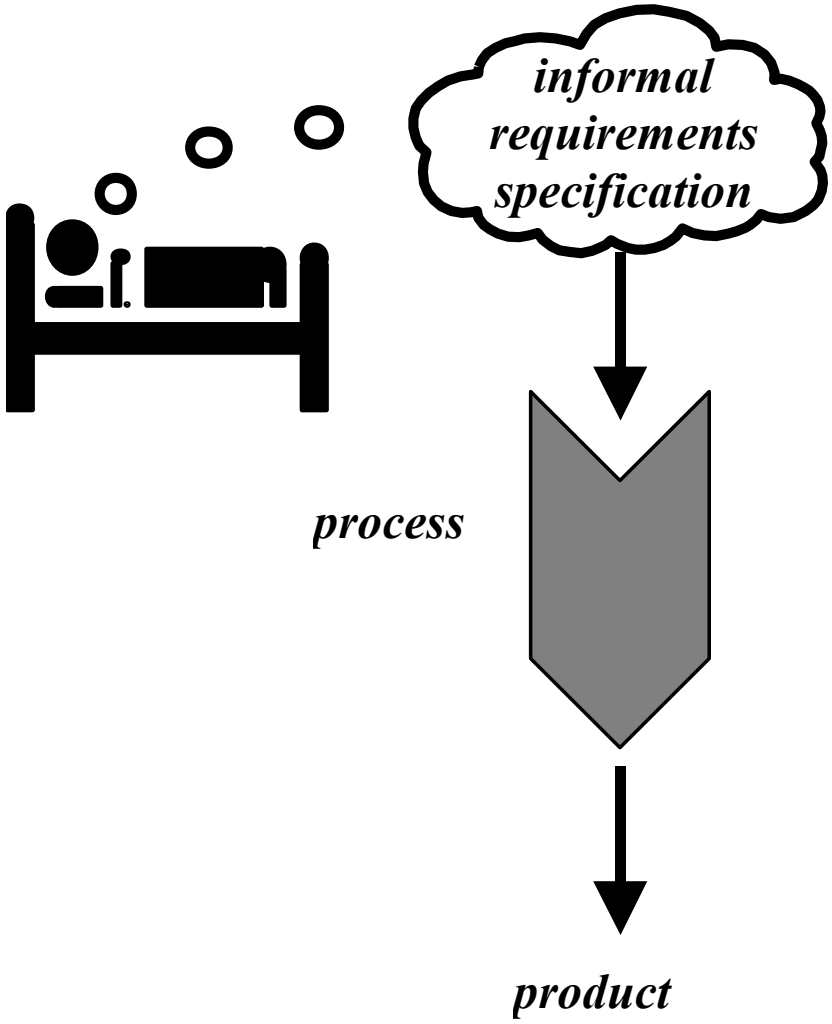
Risks of evolution

How to face evolution

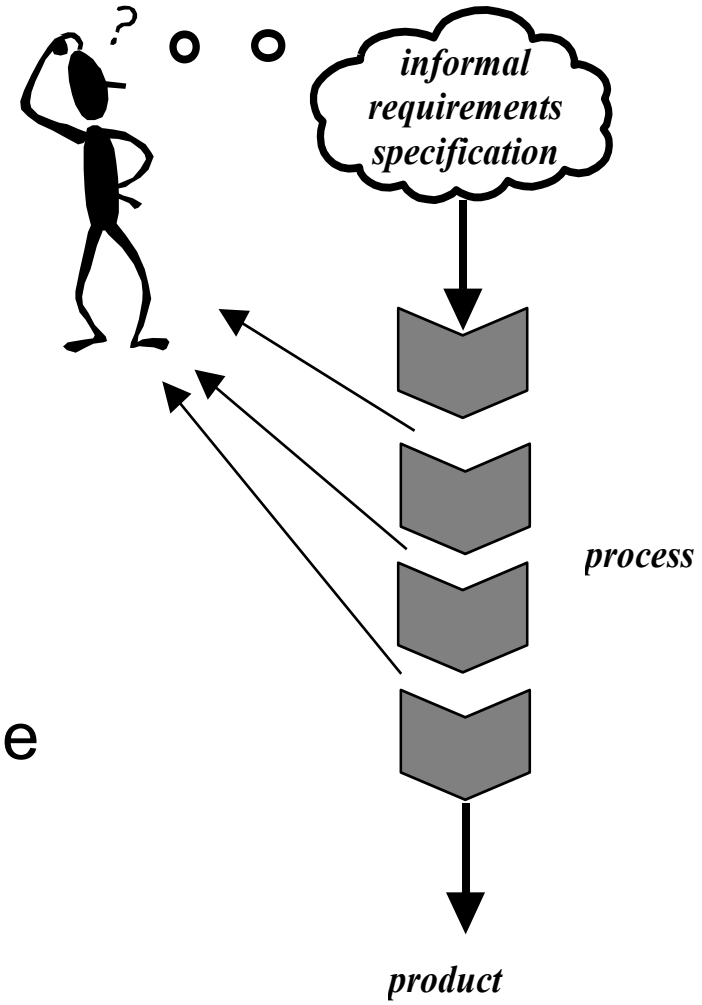
- Likely changes must be anticipated
- Software must be designed to accommodate future changes reliably and cheaply

This is one of the main goals of software engineering

Waterfall is “black box”

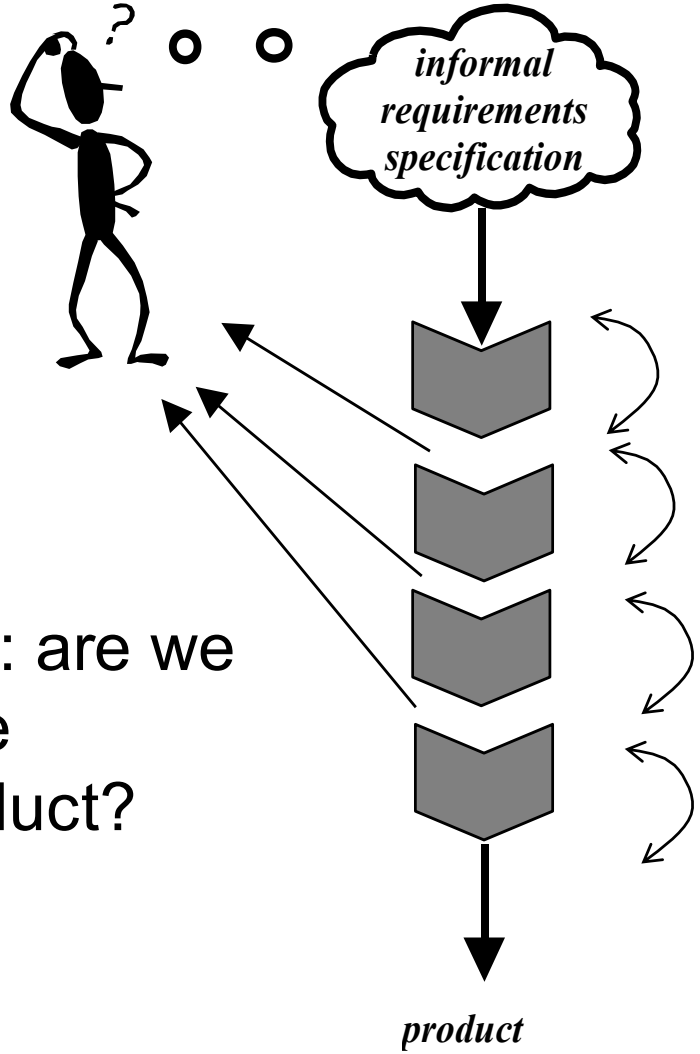


Need for transparency



- Transparency allows early check and change via feedback
- It supports flexibility

Verification and validation



- **Validation**: are we making the ”right” product?
- **Verification:** are we doing the product right?

Flexible processes

- Adapt to changes, in particular in the requirements and specification
- The idea is to have incremental processes and be able to get feedback on increments
- Exist in many forms
  - SCRUM
  - Extreme Programming
  - Incremental releases and rapid prototyping
  - …
- Also because of new deployment scenarios
  - Web-based applications
  - Mobile “apps”
  - ...


POLITECNICO
MILANO

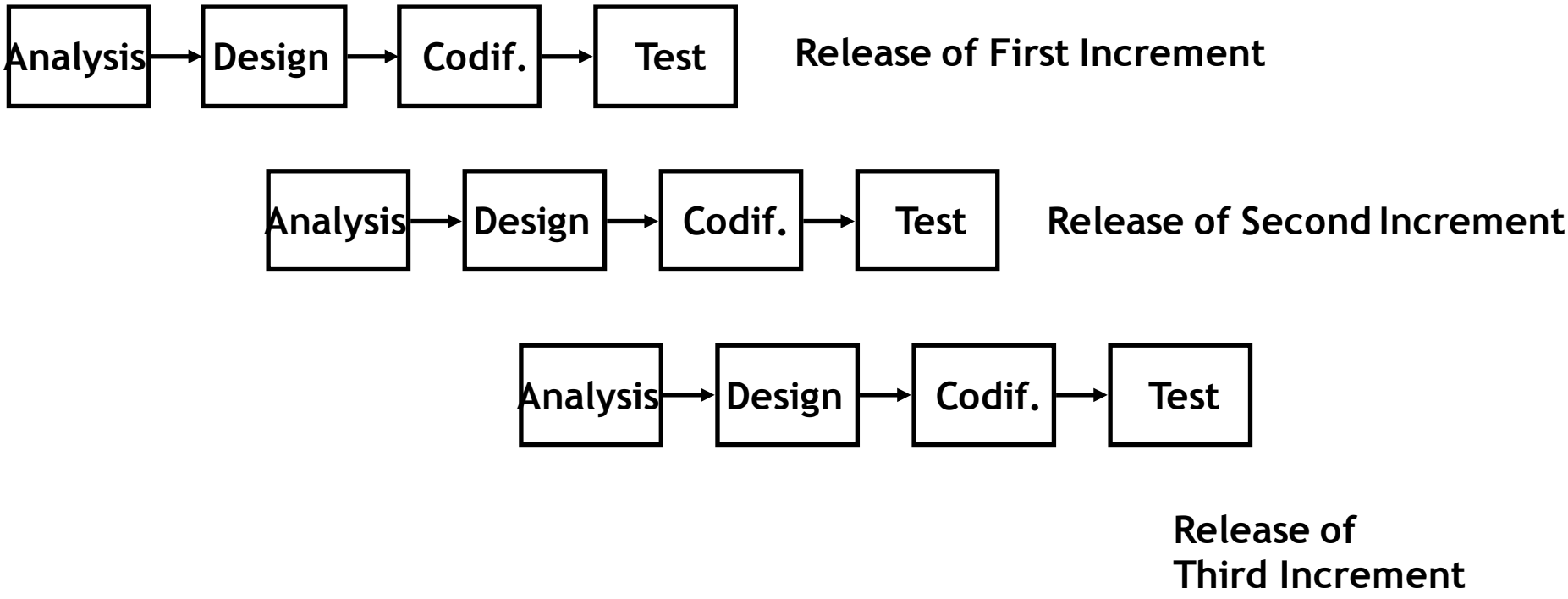
Prototyping

- A prototype is an approximate model of the application, used to get feedback or prove some concept
- “What” to prototype depends on what is critical to assess (e.g., user interface)
- Throw-away vs evolutionary prototype

Incremental delivery

- Early subset, early delivery, … early feedback
- Start from critical subsets, on which feedback is required from customer

Incremental delivery



New products and lifecycle

- Incrementality even more important for new types of product
- Beta versions available for try-out
- The network as a distribution medium and a showcase

A new issue: the Dev & Ops dichotomy

Dev focuses on

- Developing features
- Changing requirements
- Releasing products

Ops focuses on

- Offering services
- Providing guarantees and stability

"10 days after successful deployment of last release, the system is overloaded"

"The system works correctly in the development machine but it does not on the operation machine"

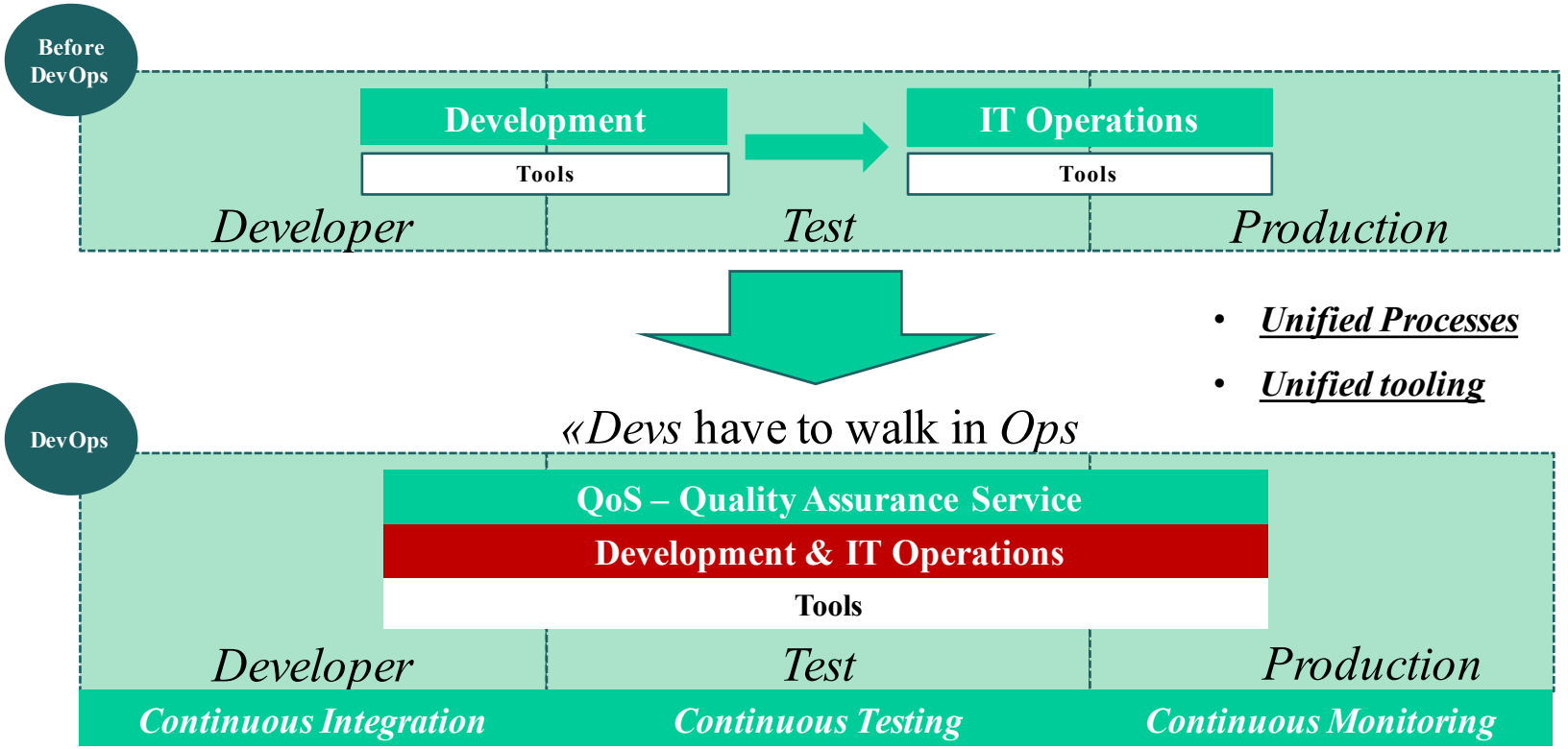
Who is guilty, Dev or Ops???

DevOps

- Bridges the gap between development and operations
- Creates a collaborative mindset where a single team performs Dev and Ops

  Of course, the team will contain differentiated competences!
- Requires
  - Culture
  - Automation
  - Metrics
  - Sharing

DevOps: Process alignment



Open source development: goals

- To produce software freely available to anybody as a source code.
  - Different variations and interpretations, though.
- Developed by a high number of developers, possibly distributed worldwide.
- No direct economic reward from development activities.
- Each developer volunteers to develop some portion of the code.

Open source development

- Distributed development:
  - Communication based on Internet tools: email, bulletin boards, discussion groups, newslists.
  - Configuration management accomplished through a centralized site running CVS.
- A main designer (or a limited number of people) manages the project:
  - Definition of the project strategy.
  - Development of core components.
  - Release management and control.
- See Linux.

Apache

- The process:
  - Architectural definition (carried out by a single person).
  - Porting of the features offered by NCSA httpd.
  - Development of new features (6 months, 4-6 people).
  - Evolution ...

Apache

- Evolution
  - Core developers (Apache Group, AG)
    - The vote for the inclusion of new features or changes to the existing code.
    - The accept new members into the group (8 founding members in ’95, 25 in 2000).
  - Development activities carried out individually:
    - Identification of the development need (e.g., fixing a specific bug). Accomplished through mailing list, problem report sent through email, …
    - Identification of a volunteer.
    - Development of new code that is submitted to the AG.
    - AG: accomplishes a review of the submitted code and decides whether to include it or not in the official version.

Apache

- Release
  - One of the *core developers* takes the role of *release manager.*
  - Release manager:
    - Evaluates the stability of the present release.
    - Identifies problems that can make the release unfeasible.
    - Manages the modification of the product repository.
  - “shake the tree before raking up the leaves”
    - Resolution of open problems.
    - Inclusion of proposed modifications.

Observations

- There is a "*core team*" of developers (10-15).
- Key roles played by a limited number of people.
- Reaction time are very quick:
- Very often, users and developers are the same:
  - **Pro**: deep knowledge of the application domain.
  - **Cons**: limited applicability (so far, software open source = system software).

Advantages of open source compared to closed source development

- … to be written in class all together