Introduction to Software Engineering

Introduction

Philippe Lalanda

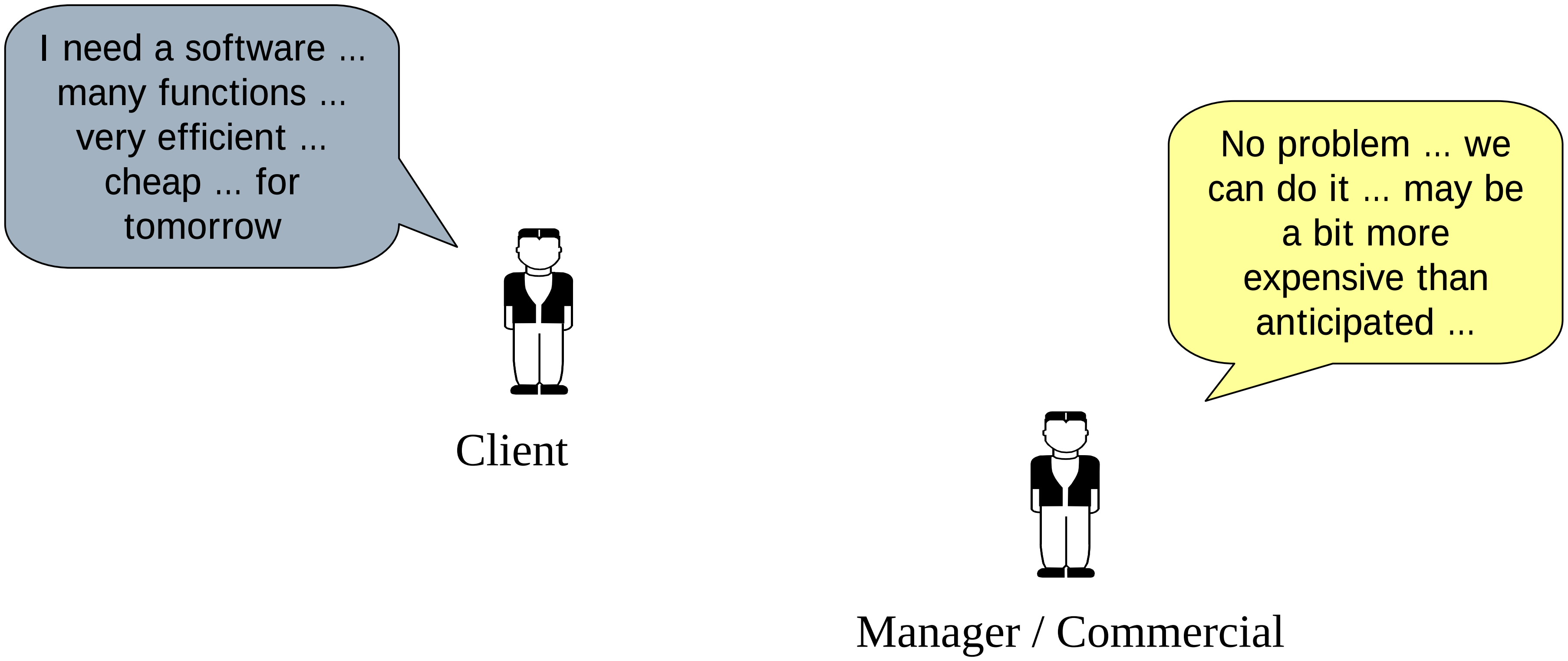
Philippe.lalanda@imag.fr

http://membres-liglab.imag.fr/lalanda/

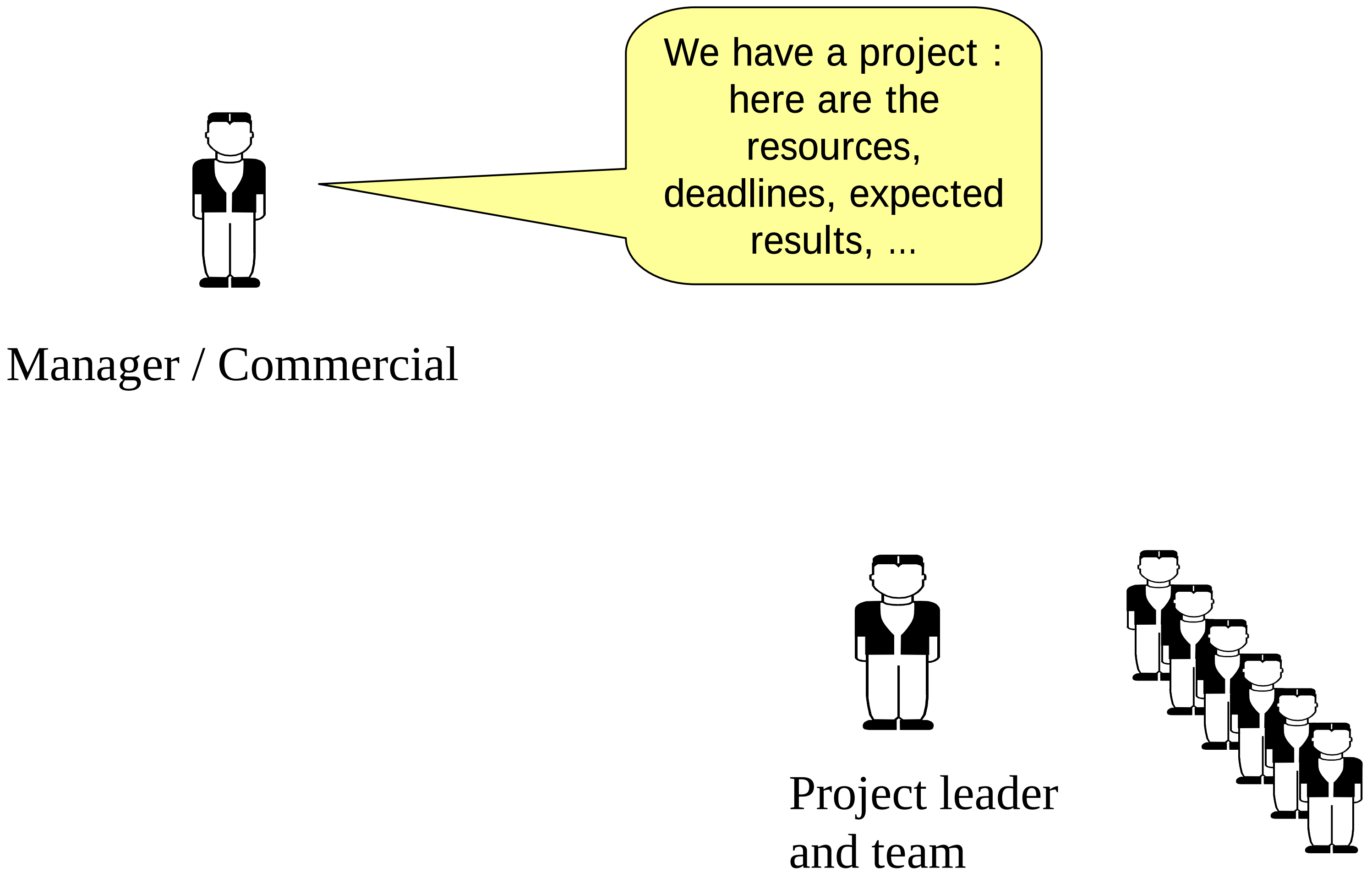
Outline

- A short, compelling story
- Historical perspective
- Software characteristics
- Software project
- Software engineering
- Conclusion

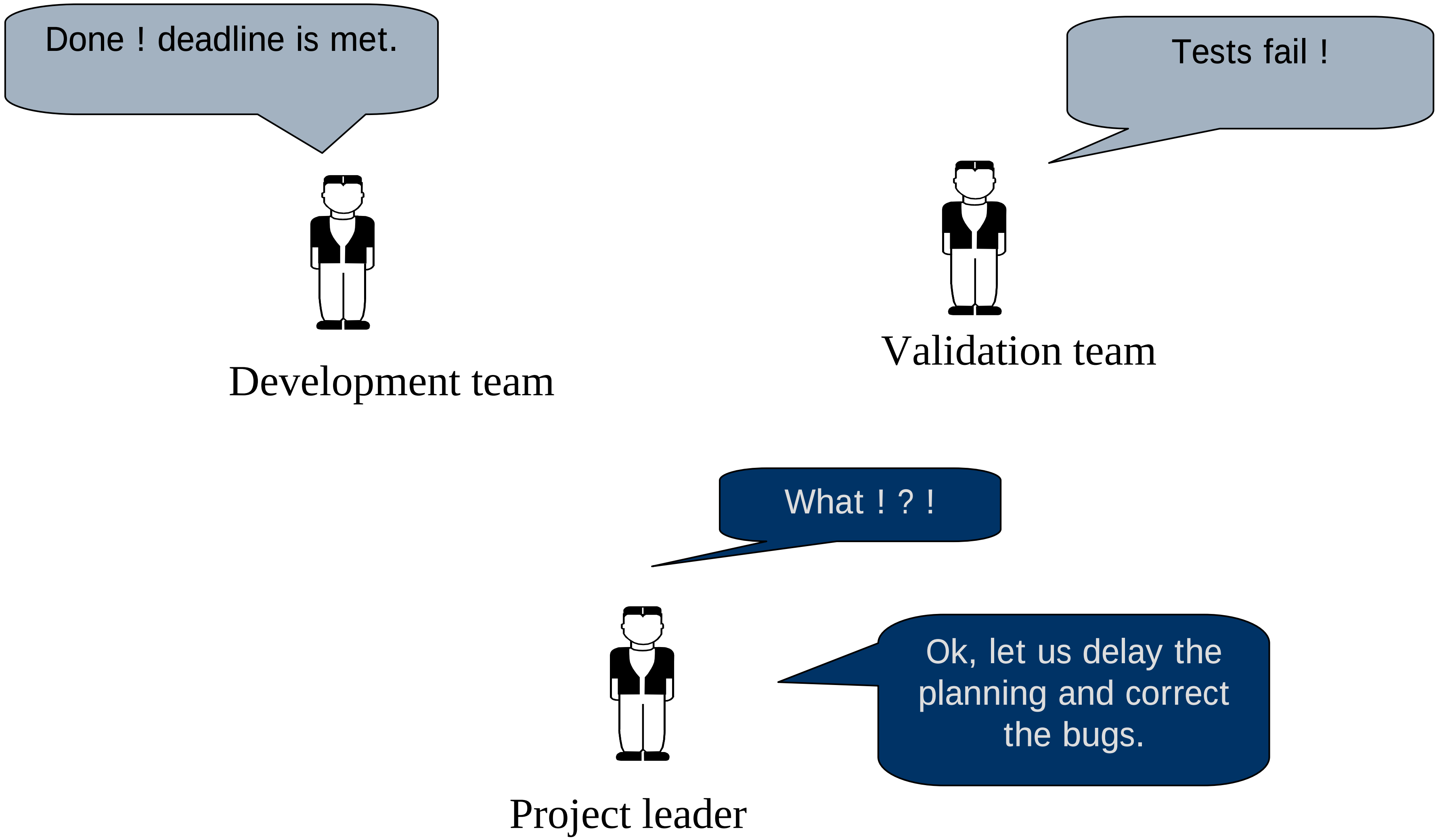
The same old story


I need a software ... many functions ... very efficient ... cheap ... for tomorrow
Client
No problem ... we can do it ... may be a bit more expensive than anticipated ...
Manager / Commercial

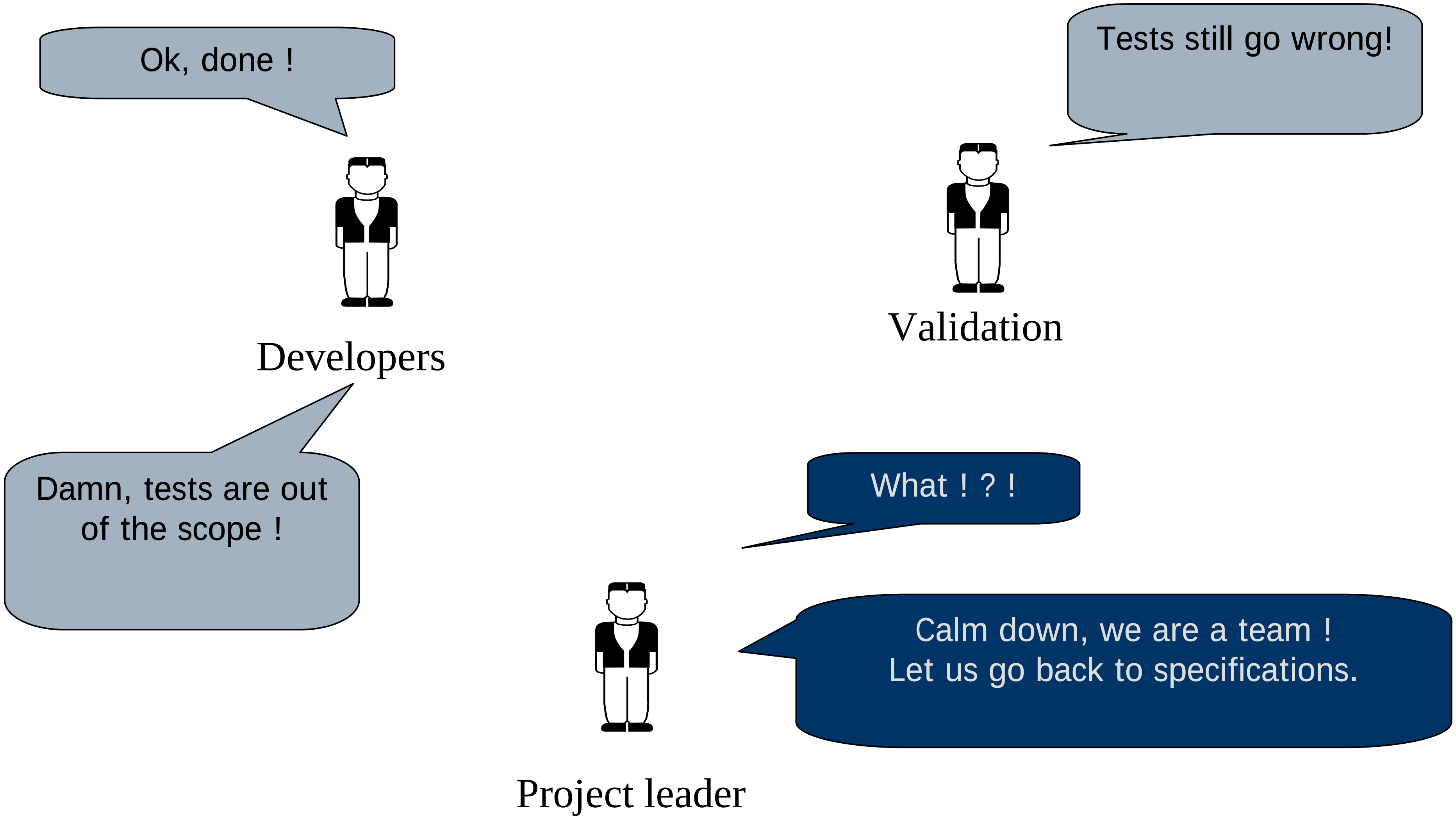
The same old story


We have a project : here are the resources, deadlines, expected results, ...
Manager / Commercial
Project leader and team

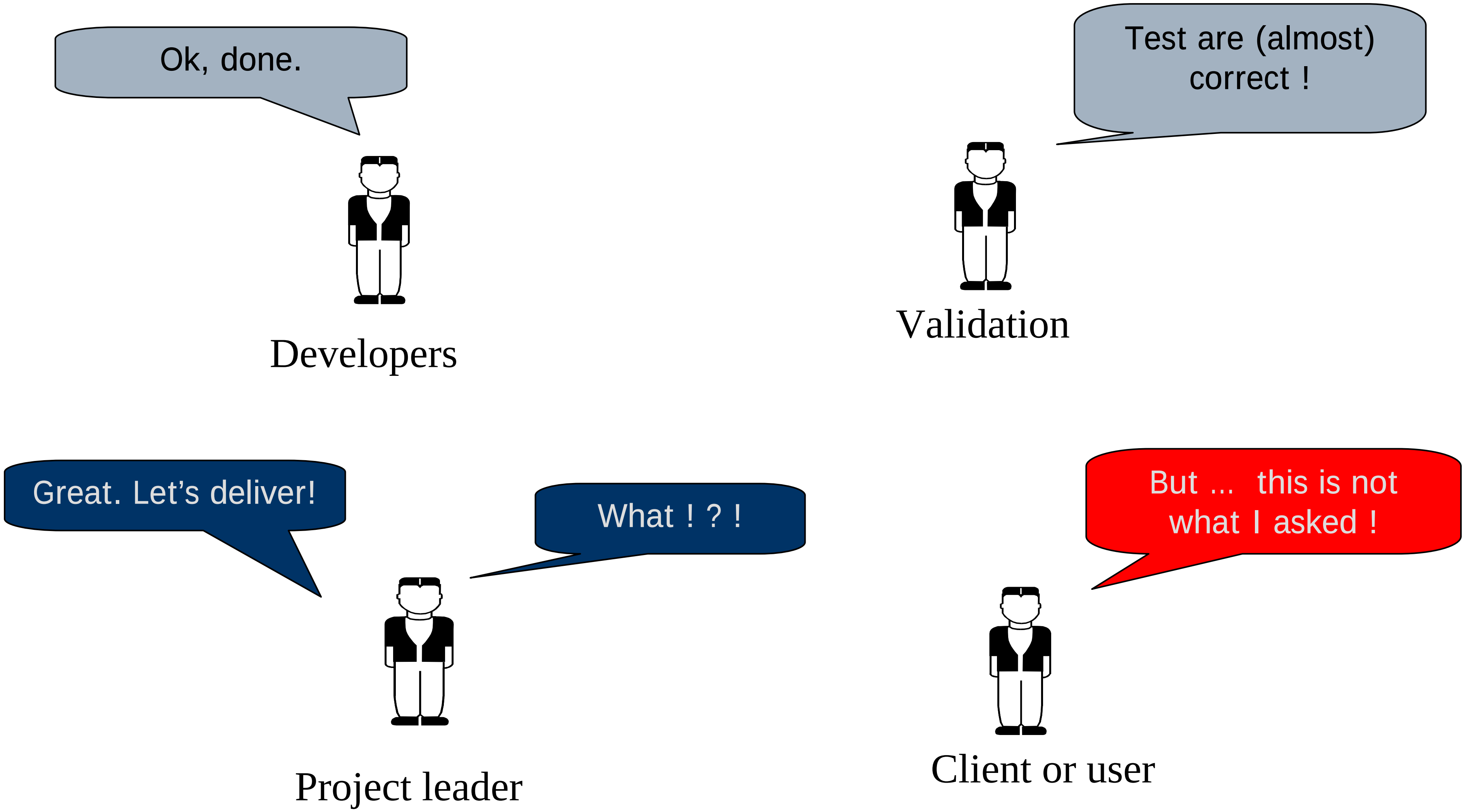
The same old story



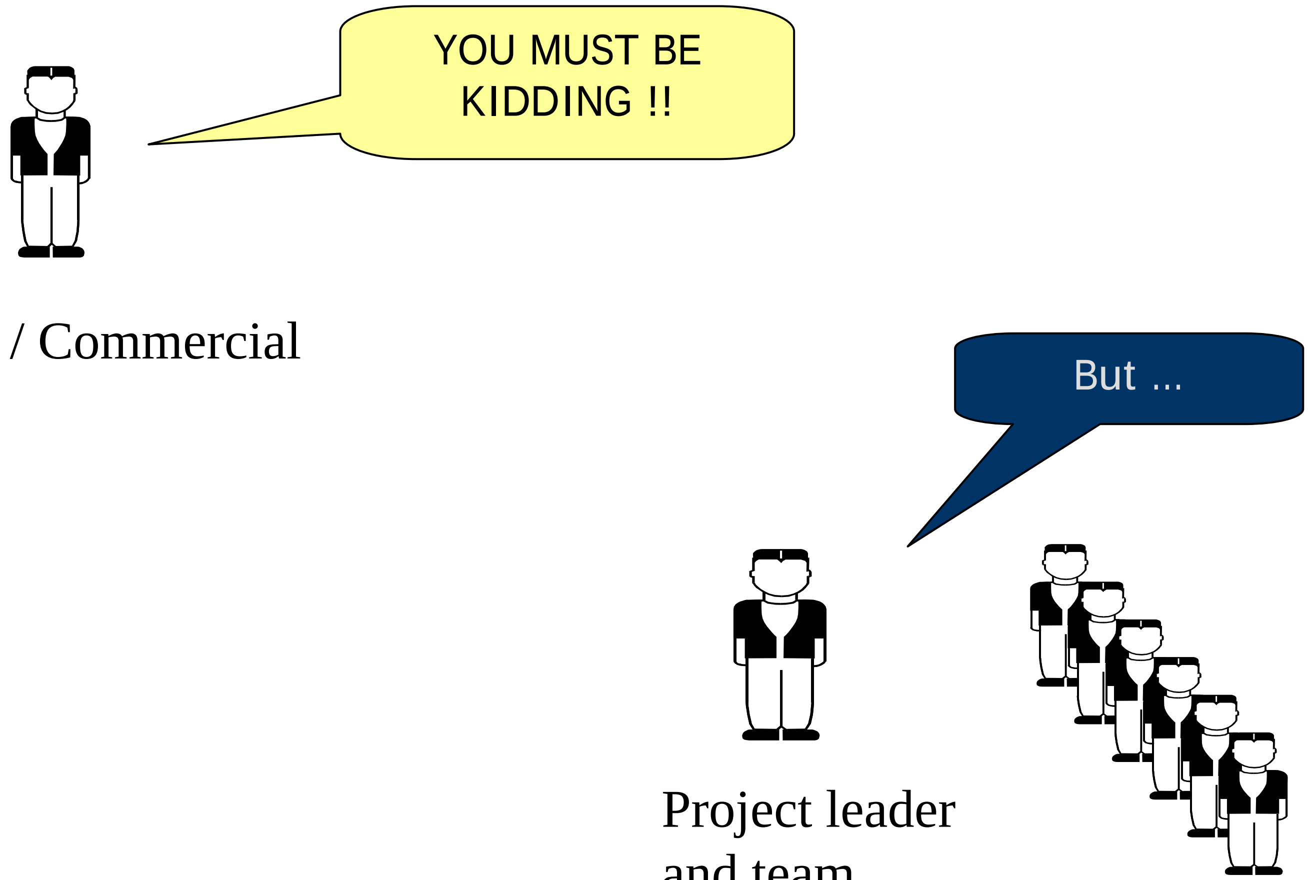
The same old story


Ok, done !
Developers
Tests still go wrong!
Validation
Damn, tests are out of the scope !
What ! ? !
Calm down, we are a team ! Let us go back to specifications.
Project leader

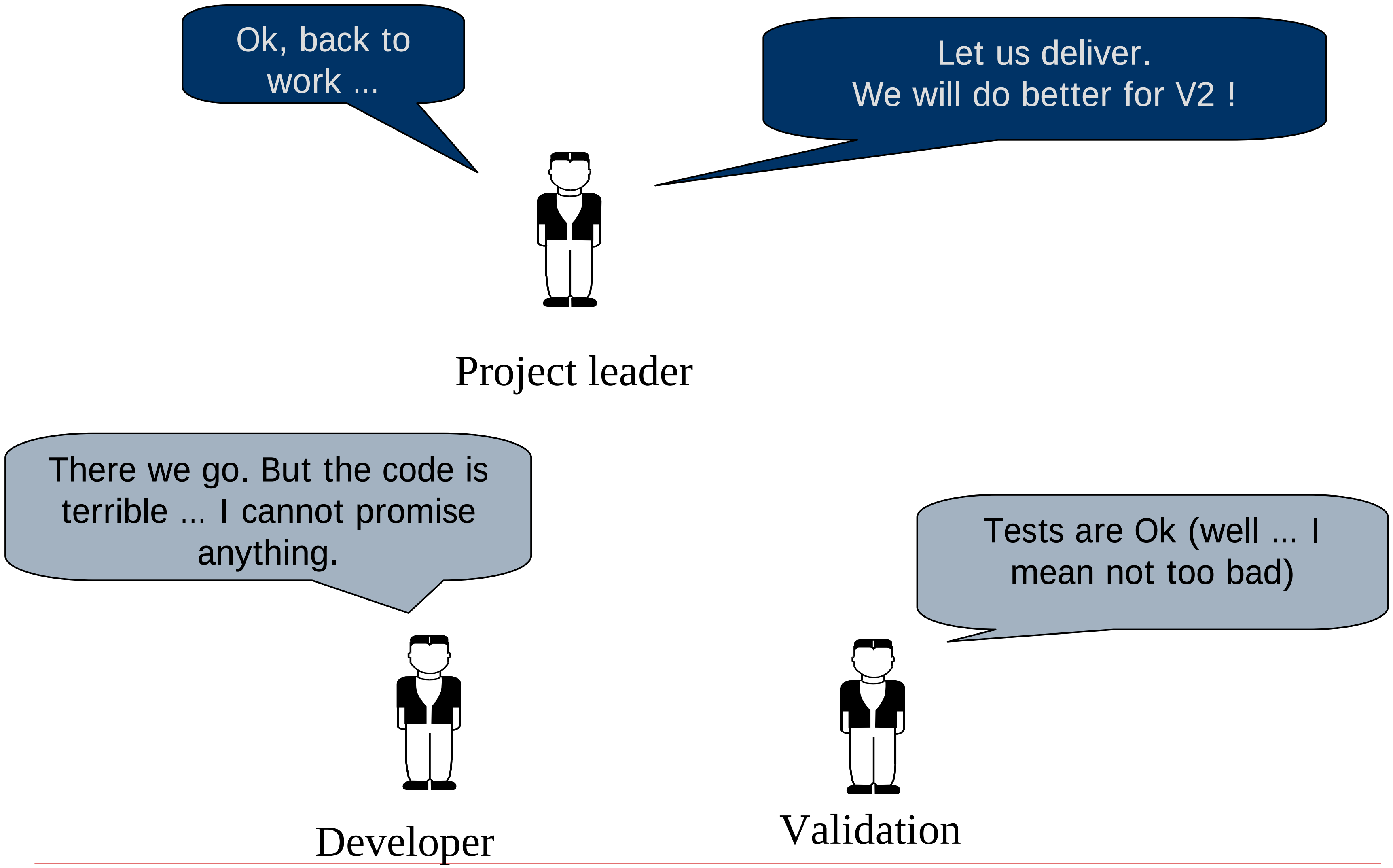
The same old story



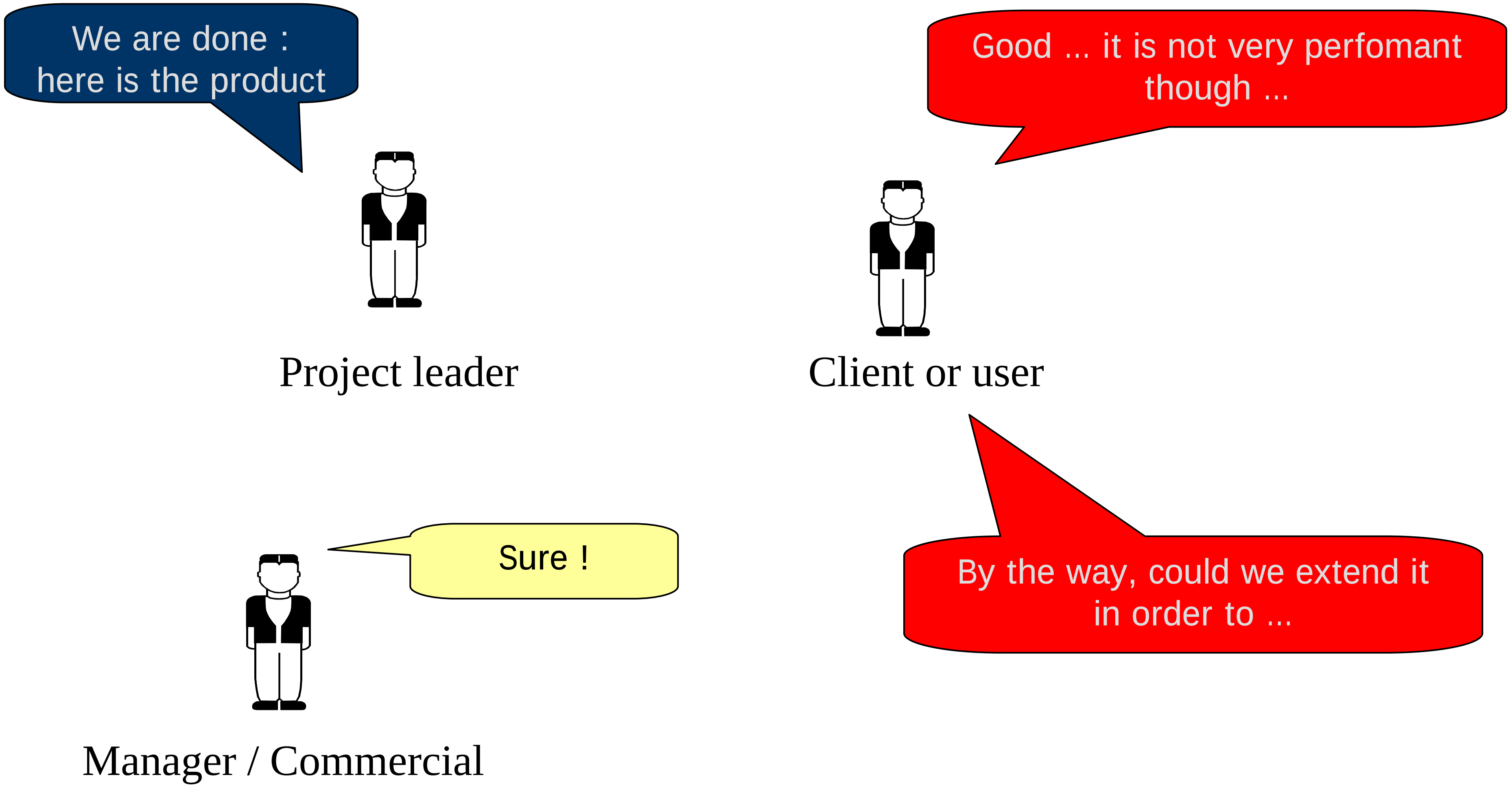
The same old story



The same old story


Ok, back to work ...
Let us deliver. We will do better for V2 !
Project leader
There we go. But the code is terrible ... I cannot promise anything.
Tests are Ok (well ... I mean not too bad)
Developer
Validation

The same old story


We are done : here is the product
Project leader
Good … it is not very perfomant though …
Client or user
Sure !
By the way, could we extend it in order to …
Manager / Commercial

Main issues

- This illustrates many issues of software projects
  - Bad estimates
  - Lack of support from hierarchy
  - Lack of experience from project leader
  - Communication issues
  - Human issues
  - Unclear requirements
  - Lack of users involvement
  - Lack of specification, tests, …

Purpose of this class

- Introduction to SE: an attempt to apply engineering principles to software development
  - Study of software processes
  - Development principles
  - Techniques and notations

- Not a usual course
  - No language
  - No BD, …

Outline

- A short, compelling story
- Historical perspective
- Software characteristics
- Software project
- Software engineering
- Conclusion

The sixties

- Hardware evolution, first languages
- Emergence of a new job: programmer
  - Distinction between user and programmer
  - Distinction between specification and programming
- Few big size projects
  - In scientific organization (MIT, IBM)
  - Pulled out by small groups of experts (pioneers)
- Few problems and lot of hope

The seventies

- Major evolution in hardware
- Many big size projects
  - IBM OS360 operating system for instance
- A time of disillusion
  - Low quality, users dissatisfaction, delays and budgets are not met
  - Existing techniques do not scale, new problems are encountered
- Organization of conferences, new terms appear
  - « Software crisis »
  - « Software engineering »

Figures

- In the seventies (US study over 100 projects)
  - Delays off by 52%
  - Software budgets off by 72%
  - Hardware budgets off by 15%
  - Poor quality : 30 to 85 bugs for 1000 instructions
- In the eighties (US government survey)
  - 47% delivered, never used
  - 29% paid, not delivered
  - 19% used then modified or dropped off
  - 3% used with minor modifications
  - 2% used as is

Many problems … few solutions

- Problems identification
  - Bad understanding of the objectives
  - Communication overhead
  - Human management (lack of motivation, departures, …)
  - Technical evolution, requirements instability

- Solutions proposed over the years
  - Better project management practices
  - New programming paradigms (and languages)
  - Use of formal approaches
  - Creation of standards, norms, …

But …

- It has been acknowledged that developing software was an engineering practice
  - complex
  - to be based on repeatable, assessable techniques

- Software Engineering was born !
  - 1968 (OTAN conference)

Eighties and nineties

- Significative improvement in many dimensions
  - Better, more adapted organizations
  - Monitored process, permanent improvement
  - Clear separation between specification and modelisation
  - Structured programming
  - Users involvement

- The place of software expended notably
  - Projects got bigger and bigger (along all dimensions)
  - Problems remained !

Nineties

- CHAOS report, Standish Group
  - 31% projects are cancelled
  - 53% projects exceed allocated resources
    - Cost: 198%
    - Time: 222%
  - 16% projects go as expected
    - 61 % of these projects meet the original requirements
  - Side note: in large companies, 9% projects go as expected !

Examples

- Infamous projects
  - Interruption of ATT long distance calls ('92)
  - Lost of electronic votes ('92)
  - Driver licenses management (California DMV) after 6 years and $45 spent ('93)
  - Car plus hotel reservation (American Airline, Budget, Hilton) after $165 spent ('94)
  - Interruption of French train reservation ('94)
  - Ariane ('95)

- See http://blogs.spectrum.ieee.org/riskfactor/

More fun

- Siemens AG announced that its first quarter profits would be about $1.4 billion lower than expected. Part of the reason was, as ComputerWorldUK explains, the cancellation of a major IT contract in the UK for the Department of Work and Pensions (DWP). In April 2004, Siemens was contracted to provide a a central payment system as well as provide ongoing management and maintenance through to 2010.
- The DWP canceled the project because of some small problems with schedule and cost. The payment system was supposed to be completed by October 2006, but it slipped to December 2010.
- Project costs also increased from £90 million budget to an estimated cost of £153 million.
- Siemens also indicated that there were other problems with its IT units that are contributing to the profit warning.

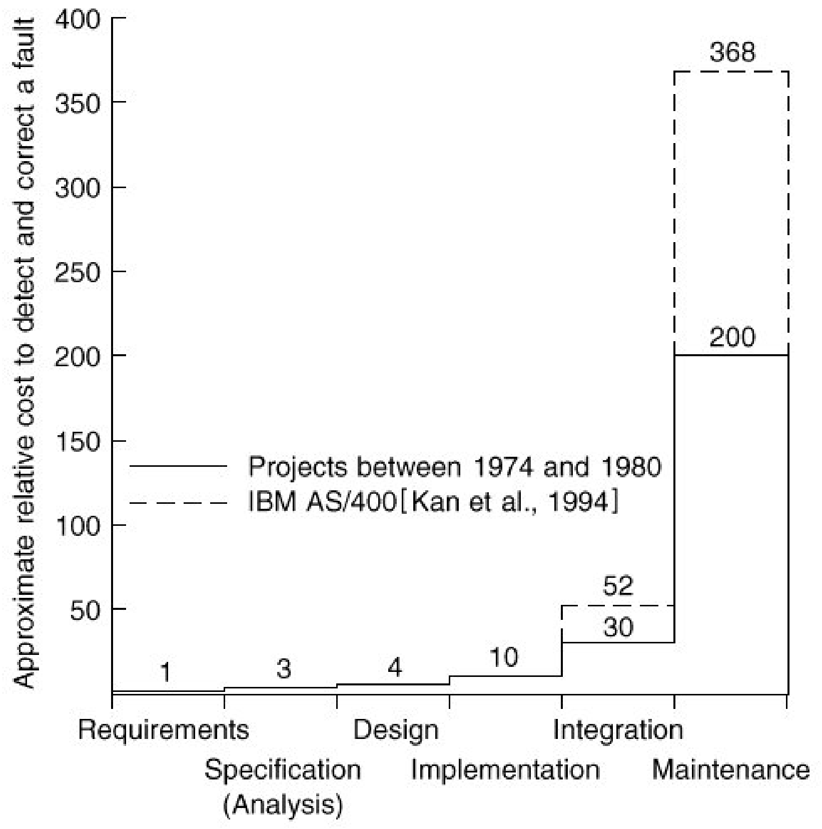
2000's

- CHAOS report, Standish Group
  - 23% projects are cancelled
  - 49% projects exceed allocated resources
    - Cost: 43%
    - Time: 63%
  - 28% projects go as expected
    - 67 % of these projects meet the original requirements
  - Encouraging improvement !

Cost

- This represents a true prejudice for companies
  - Financial lost when a project is cancelled
  - Missed opportunities
  - Missed commercial launches
  - Lack of desired software
    - Lack of productivity
    - Lack of services
    - Lack of support
  - Cost of debugging (corrective maintenance)

Cost of finding flaws late



The human factor

- Probably 90% of project failure is due to politics » (Standish Group)
  - Wrong organization
  - Fighting bosses
  - Unsupportive hierarchy
  - Univolved users
  - Lack of champion
- Need to concentrate on activities involving heavy communication

Robert Glass, 1995 and 2004

- " … In which, I question the unquestionable Standish Chaos findings"
  - Some studies are biased (based on projects in peril)
  - Some researchers take the results without assessment (to justify their work)
  - Standish is an opaque organization (the report is billed $5000)
  - The idea of failure is relative

Other projects

- Non software projects too have problems
  - Hard structures collapse
  - Delays are exceeded
    - Olympics, Tram, TGV, …
  - Costs are exceeded
    - Stade de France, TRAM, …

Outline

Software characteristics

- Software is
  - unique
  - malleable
  - intangible
  - complex and large
  - human intensive
  - weird !

Software is unique (Brooks, 1987)

- Software raises specific issues
- « accidental » issues
  - Related to technological weakness
  - This can be improved with new tools, new methods
- « essential » issues
  - Requires intelligence, creativity, time and resources
  - No silver bullet
- Please no dubious analogies …

Software is malleable

- Always good reasons to change a software
  - Changes in the environment
  - Changes in requirements
  - A software which is used implies new demands
  - Need to avoid a new development
- Related issues
  - Hard to manage side effects of a modification
  - Hard to anticipate evolution

Software is intangible

- Software is abstract and intangible : it cannot be properly viewed
- Related issues
  - Hard to understand and to make understand
  - Hard to estimate
  - Hard to manage
  - Hard to communicate

Software is complex and large

- Many kinds of projects
  - Diverse domains and environments
  - Diverse technologies and practices
- Related issues
  - Hard to stay up to date
  - Hard to reuse previous experiences
  - Hard to choose the best approach

Software is human intensive

- It relies on human skills
  - Mathematics
  - Computer science
  - Management
  - Psychology
- It implies social interactions

A weird community

- A bugged software is close to normality
  - Deployment of beta version
    - Now explicitly asked for by marketers, managers, …
  - Strong resilience to bugs
- Related issues
  - Hard to impose efficient quality politics
  - Hard to convince developers and managers

Conclusion

- Software has unique characteristics
- Specific techniques must be defined and used to deal with software projects
    - Software engineering discipline

Outline

- A short, compelling story
- Historical perspective
- Software characteristics
- Software project
- Software engineering
- Conclusion

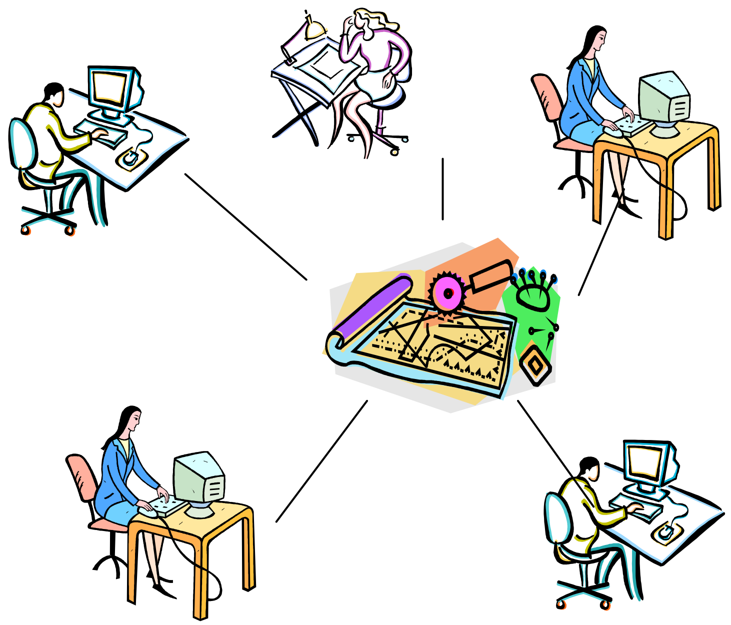
Context

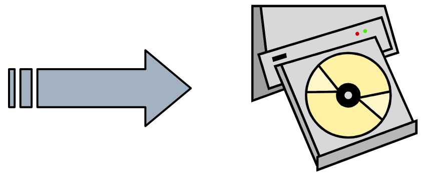
Many persons with different objectives

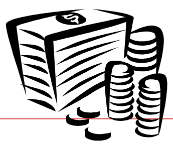
To coordinate

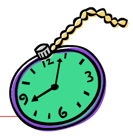
Who have to communicate

To deliver a result … meeting some constraints









Definition of a project

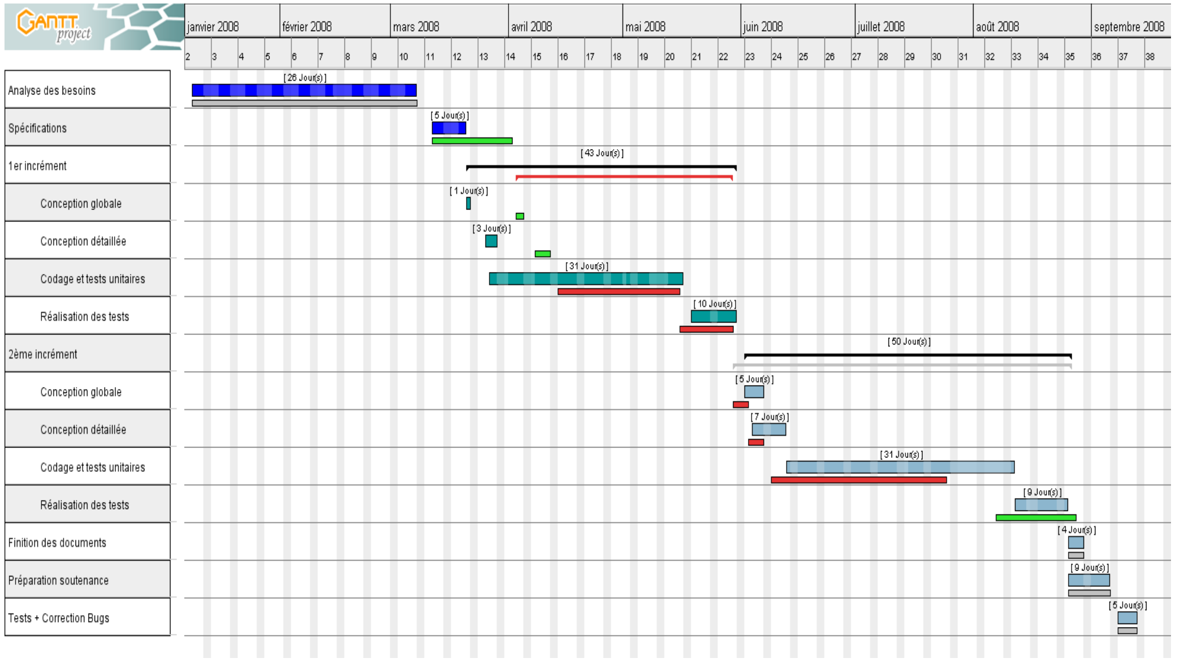
Resources

Quality

Time

Timeliness

- This implies
  - Planning (as a strategic tool)
  - Risk management



Resources

- Resources include
  - People
  - Money
- It always goes down to labor issues
  - Done internally or externally
  - Finding the good ratio is hard

Qualities

- External qualities
  - visible to the users
  - Reliability, efficiency, usability
- Internal qualities
  - concern of developers
  - they help developers achieve external qualities
  - verifiability, maintainability, extensibility, …

Some software qualities (1)

- Correctness
  - ideal quality
  - established w.r.t. the requirements specification
  - absolute
- Reliability
  - statistical property
  - probability that software will operate as expected over a given period of time
  - relative

Some software qualities (2)

- Robustness
  - “reasonable” behavior in unforeseen circumstances
  - subjective
  - a specified requirement is an issue of correctness
  - an unspecified requirement is an issue of robustness
- Usability
  - ability of end-users to easily use software
  - extremely subjective

Some software qualities (3)

- Understandability
  - ability of developers to easily understand produced artifacts
  - internal product quality
  - subjective
- Verifiability
  - ease of establishing desired properties
  - performed by formal analysis or testing
  - internal quality

Some software qualities (4)

- Performance
  - not efficiency
  - assessable by measurement, analysis, and simulation
- Evolvability
  - ability to add or modify functionality
  - addresses adaptive and perfective maintenance
  - problem: evolution of implementation is ***too*** easy
  - evolution should start at requirements or design

Some software qualities (5)

- Reusability
  - ability to construct new software from existing pieces
  - must be planned for
  - occurs at all levels: from people to process, from requirements to code
- Interoperability
  - ability of software (sub)systems to cooperate with others
  - easily integrable into larger systems
  - common techniques include APIs, plug-in protocols, etc

Some software qualities (6)

- Heterogeneity
  - ability to compose a system from pieces developed in multiple programming languages, on multiple platforms, by multiple developers, etc.
  - necessitated by reuse
  - goal of component-based development
- Portability
  - ability to execute in new environments with minimal effort
  - may be planned for by isolating environment-dependent components
  - necessitated by the emergence of highly-distributed systems (e.g., the Internet)
  - an aspect of heterogeneity

Assessing software qualities

- Qualities must be measurable
- Measurement requires that qualities be precisely defined
- Improvement requires accurate measurement
- Currently most qualities are informally defined and are
- difficult to assess

Outline

Software engineering (Sommerville)

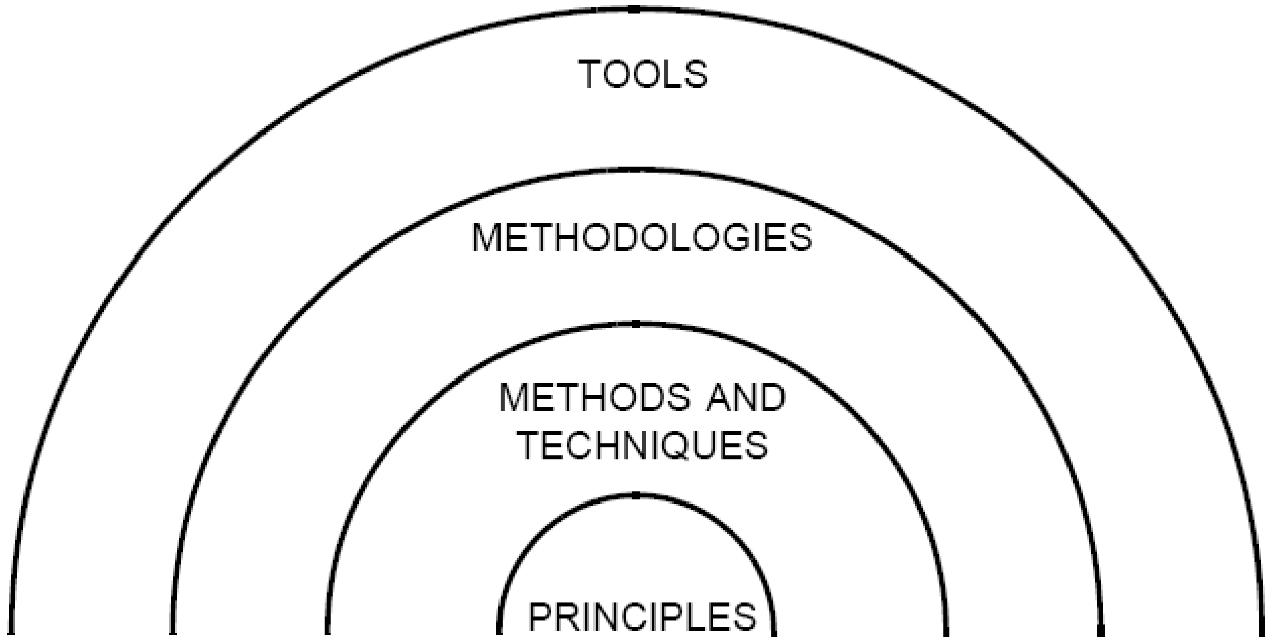
- An engineering discipline concerned with all aspects of software production
  - from early stages of specification
  - to maintaining the system after it has gone into use

- The purpose is to rigorously applies methods and techniques to produce timely, quality, satisfying software

Rigor

- As any engineering field, software engineering needs rigor
  - Definition of processes, techniques, methods
  - Definition of associated documents, deadlines
  - Validation
  - Professional attitude
- Side notes
  - Rigor does not kill creativity
  - Rigor is not equal to mathematical techniques

From principles to tools (Ghezzi, 93)



Rational Rose

UML

Object

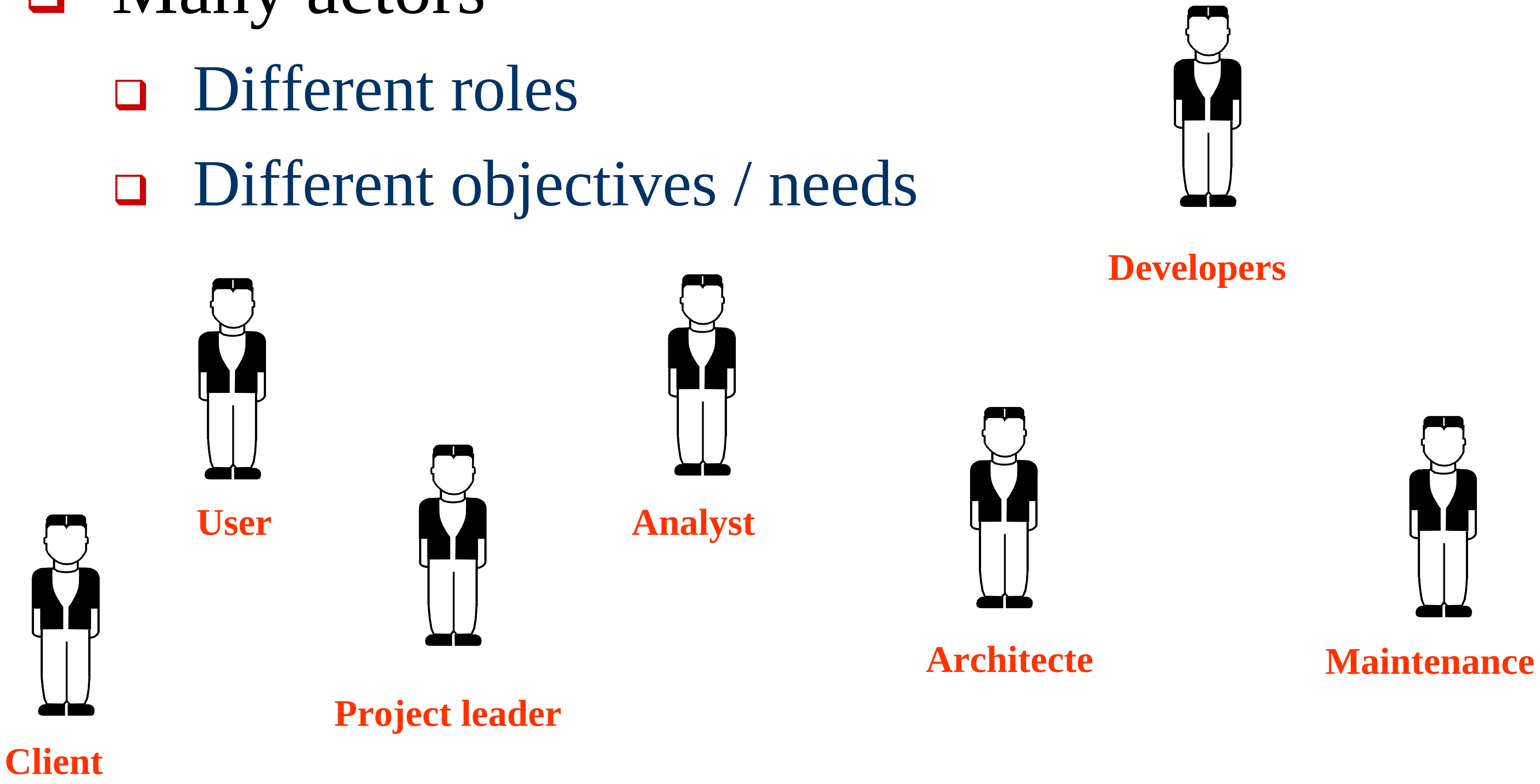
Encapsulation, hidding, …

SE outcome

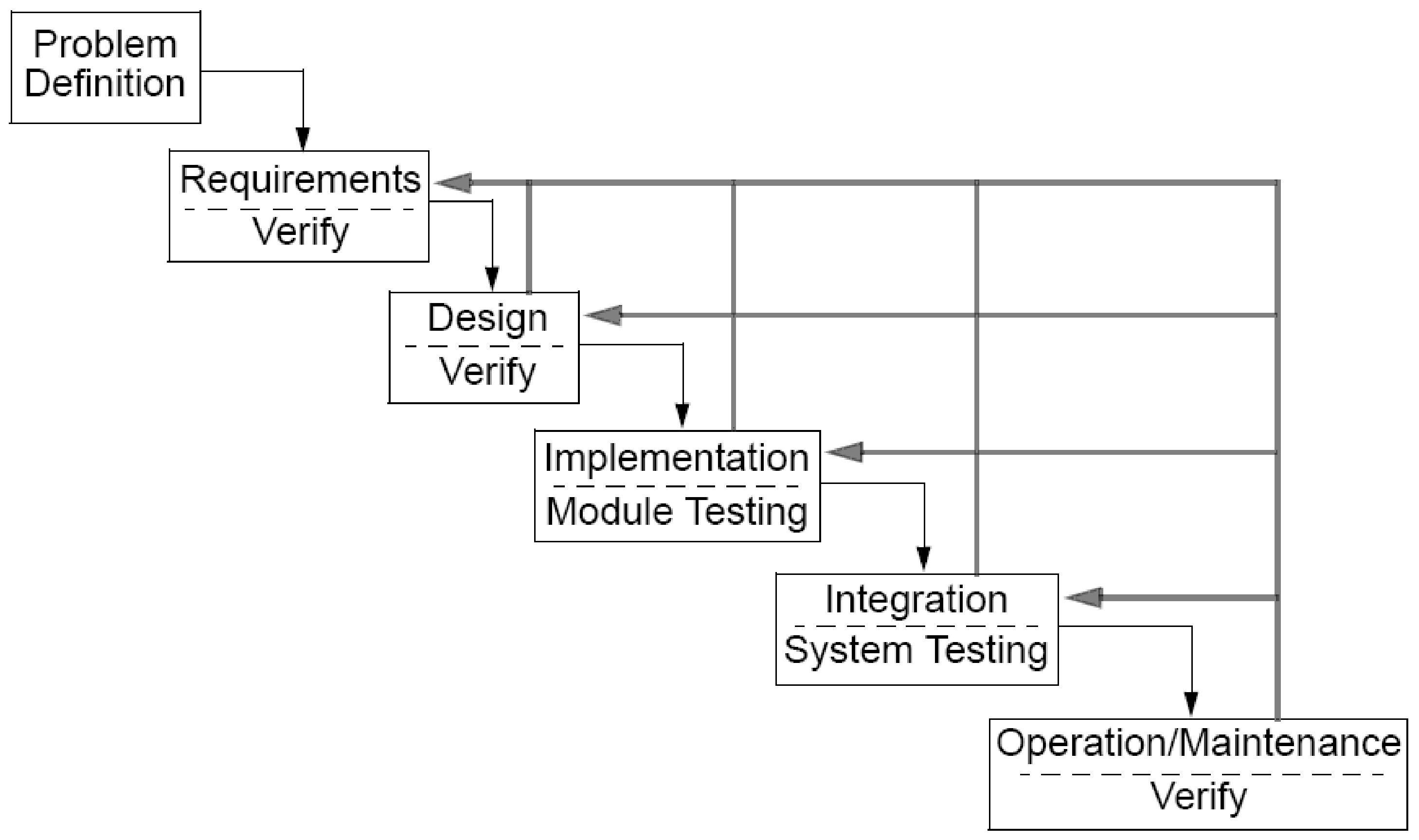
- Separation of activities and roles
  - Techniques / methods for each activity
  - Software processes
- Programming languages
  - Structured, object, functional, …
- Methodologies
  - Functional, object (UML), …
- Software management
  - Planning, risk management, outsourcing management
  - Human management
- Software improvement methods
- …

Actors / roles

- Many actors
  - Different roles
  - Different objectives / needs

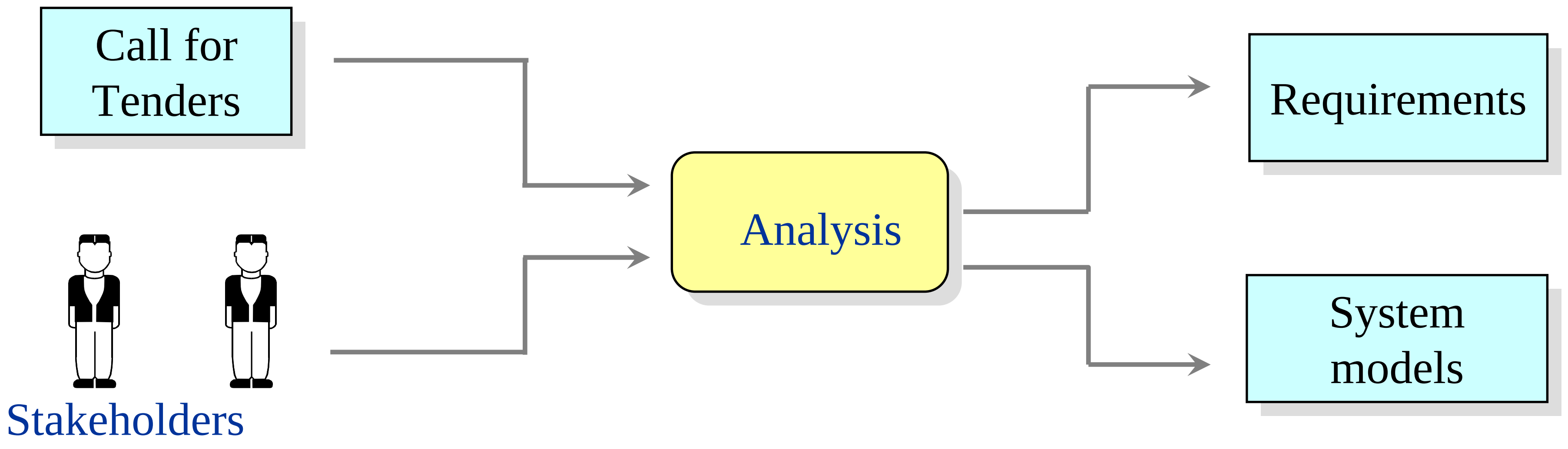


Activities


Problem Definition
Requirements
Verify
Design
Verify
Implementation
Module Testing
Integration
System Testing
Operation/Maintenance
Verify

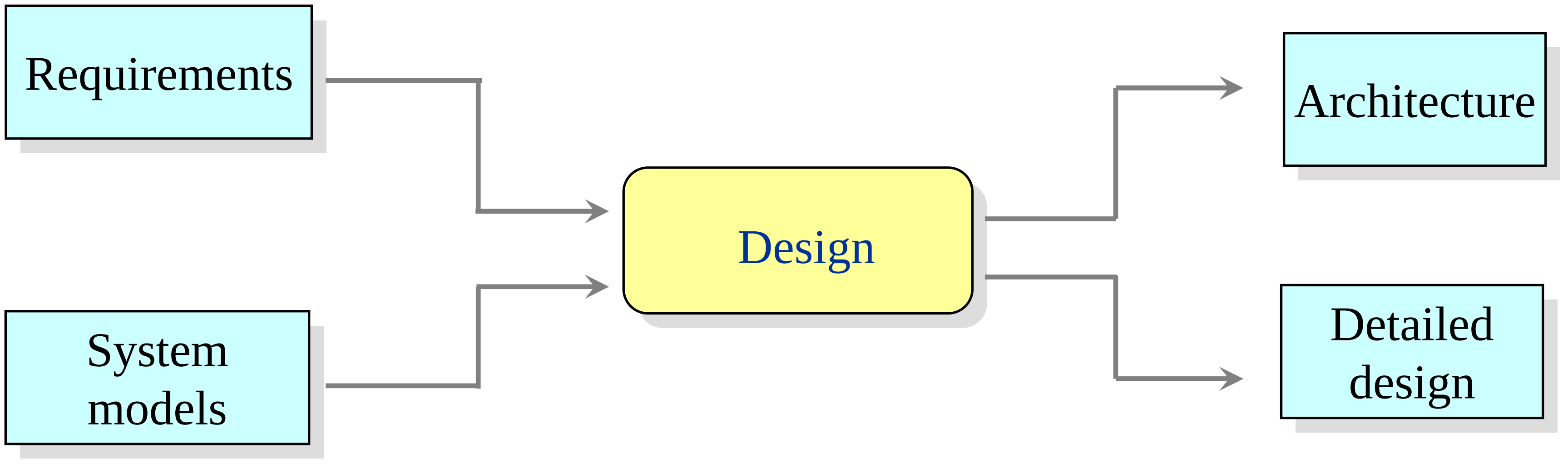
Requirements

- Objectives
  - Identify what the client wants and his constraints
  - Specify these requirements



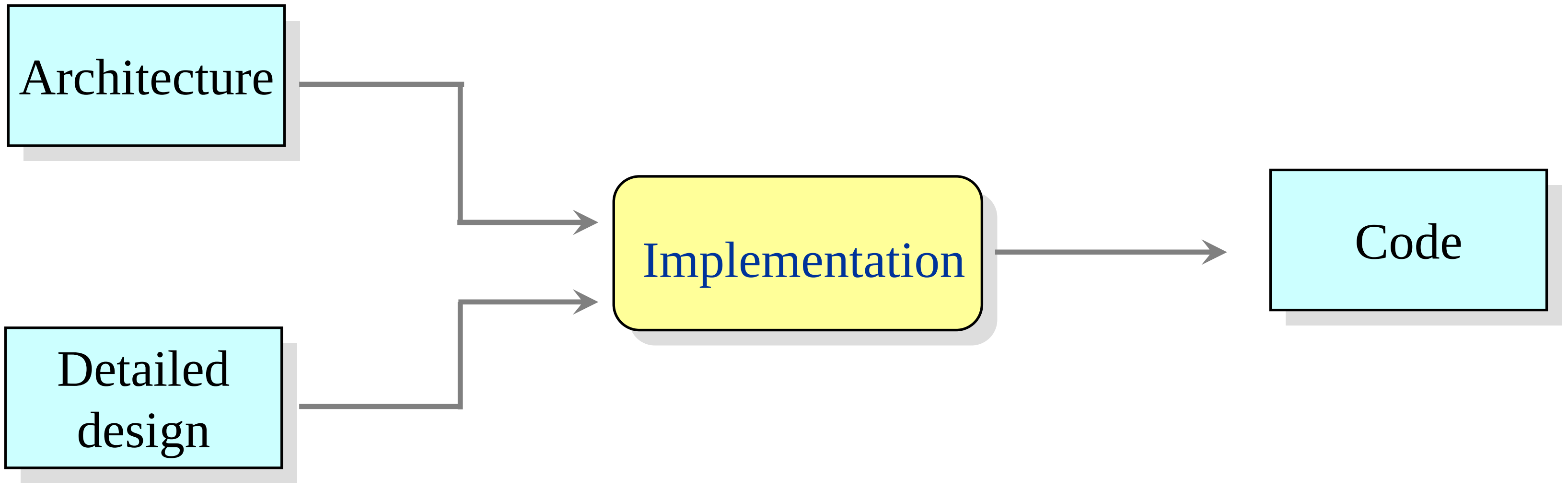
Design

- Objectives
  - Definition of a logical organization for the code
  - Provide a solution to the problem stated at analysis



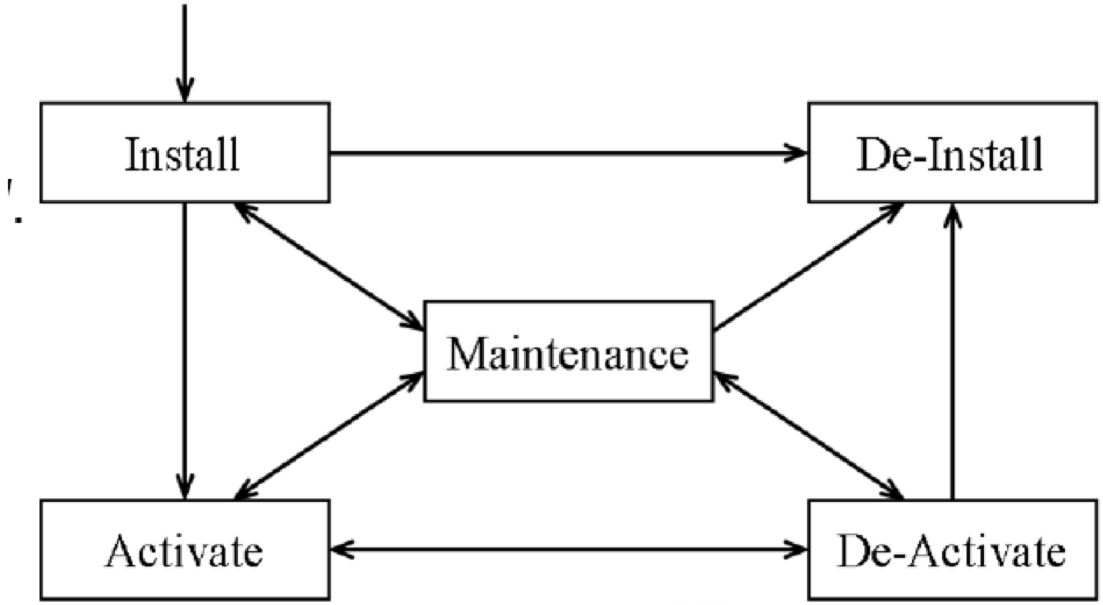
Implementation

- Objectives
  - Implement modules; verify that they meet their spec
  - combine modules according to the design



Deployment

- Objectives
  - Install (de-install)
  - Activate (de-activate)
  - Retire



Maintenance

- Objectives
  - maintain software during/after user operation
  - determine whether the product still functions correctly

- Types of maintenance
  - Corrective
  - Predictive
  - Adaptative
  - evolutive

Side note

- Most techniques rely on
  - Separation of concern
  - Abstraction and modularity
  - Models

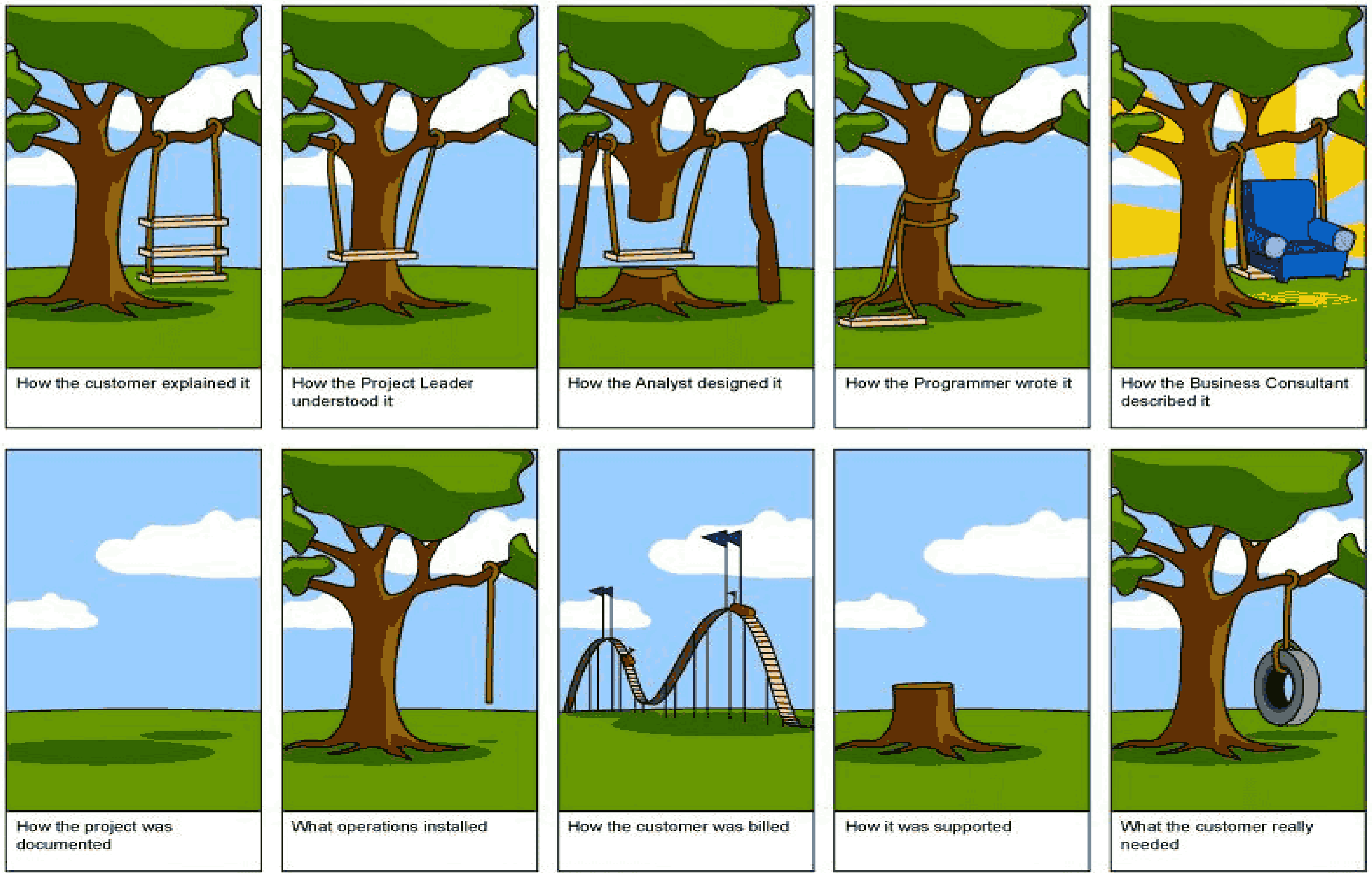
Outline

- A short, compelling story
- Historical perspective
- Software Project
- Software engineering
- Conclusion

Software programming and SE

▪ single developer	▪ developer teams ▪ multiple roles
▪ "toy" applications	▪ complex systems
▪ short lifespan	▪ long, indefinite lifespan
▪ single or few stakeholders ▫ developer = user	▪ multiple stakeholders ▫ developer ≠ user ▫ user ≠ customer
▪ one-of-a-kind systems	▪ system families
▪ built from scratch	▪ reuse to amortize costs
▪ minimal maintenance	▪ maintenance is 60+% of total development costs

Conclusion !


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed