Introduction to Software Engineering

Introduction

Philippe Lalanda

http://membres-liglab.imag.fr/lalanda/ or Google!

Miscellaneous - 1

- Organization
  - 9 lectures
  - 10 exercises sets

- Lectures and exercises sets on my page
  - http://membres-liglab.imag.fr/lalanda

Miscellaneous - 2

- Exam
  - mid-term exam – 30%
  - Final exam (multiple choice test) – 70%

- Students welcome by appointment
  - philippe.lalanda@imag.fr

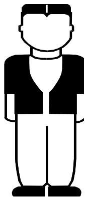
- Be on time!
  - Late students not allowed

Outline

- A short, compelling story
- Historical perspective
- Why software is hard
- Software expected qualities
- Software engineering
- Software engineering activities
- Conclusion

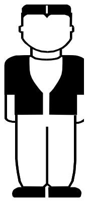
The same old story

I need a software … simply put, it must provide the following services… very efficient … cheap … asap



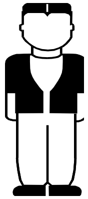
Client

No problem … we can do it … may be a bit more expensive than you wish …



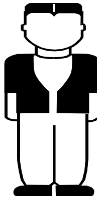
Manager / Commercial

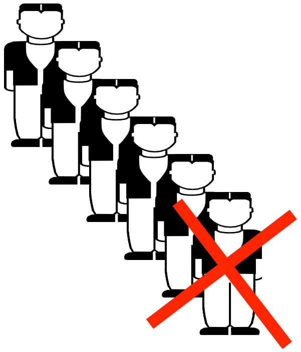
The same old story



We have a project : here are the resources, deadlines, expected results, …

Manager / Commercial

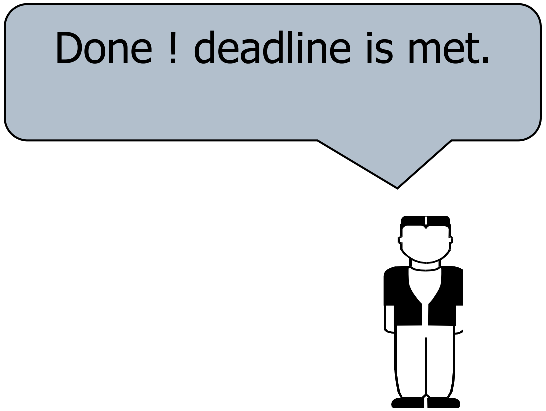




Project leader and team

No control!

The same old story



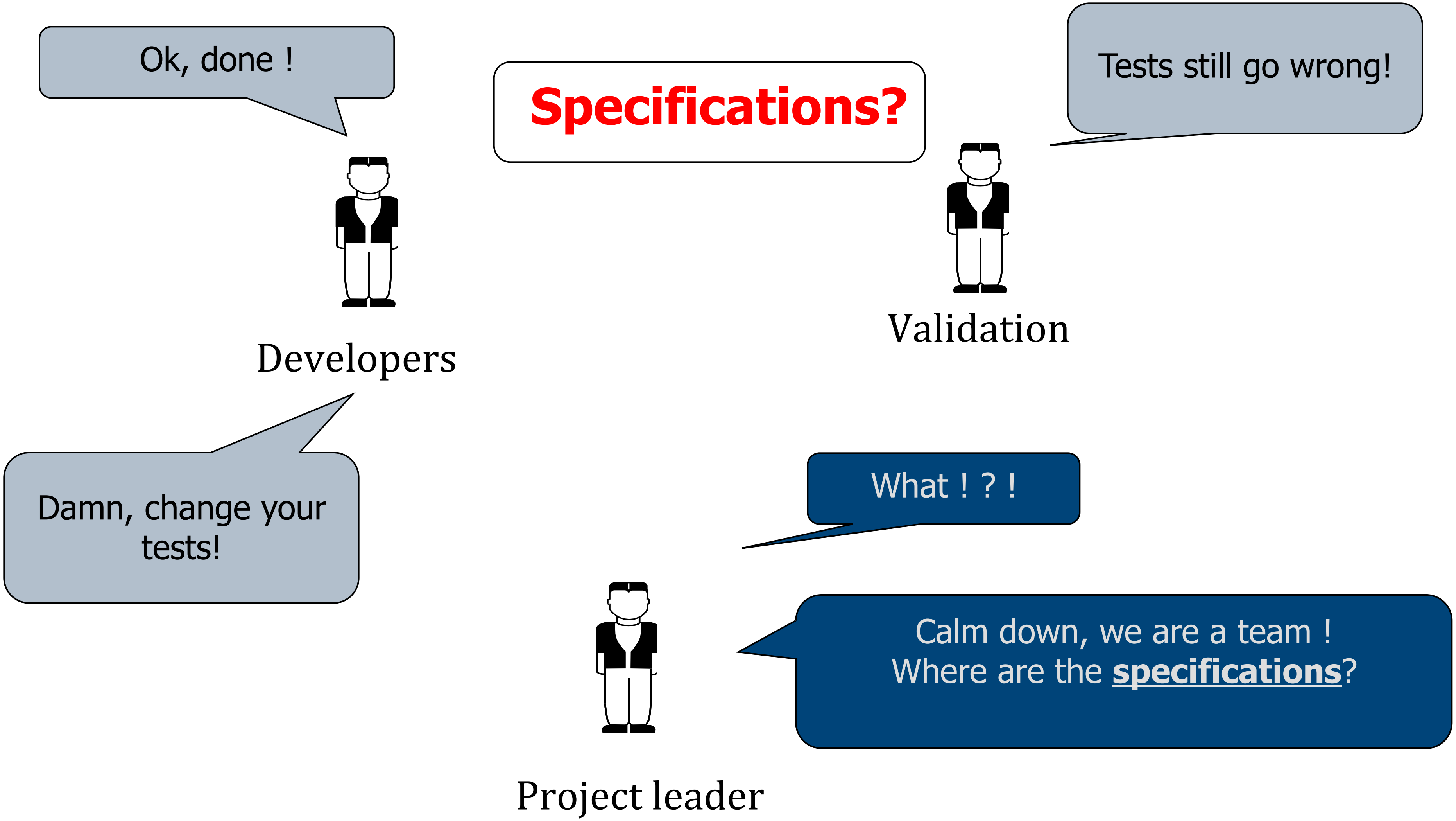
Development team

Validation team

What ! ? !
Ok, let us delay the planning and fix bugs.

Project leader

The same old story



Ok, done !

Specifications?

Tests still go wrong!

Developers

Validation

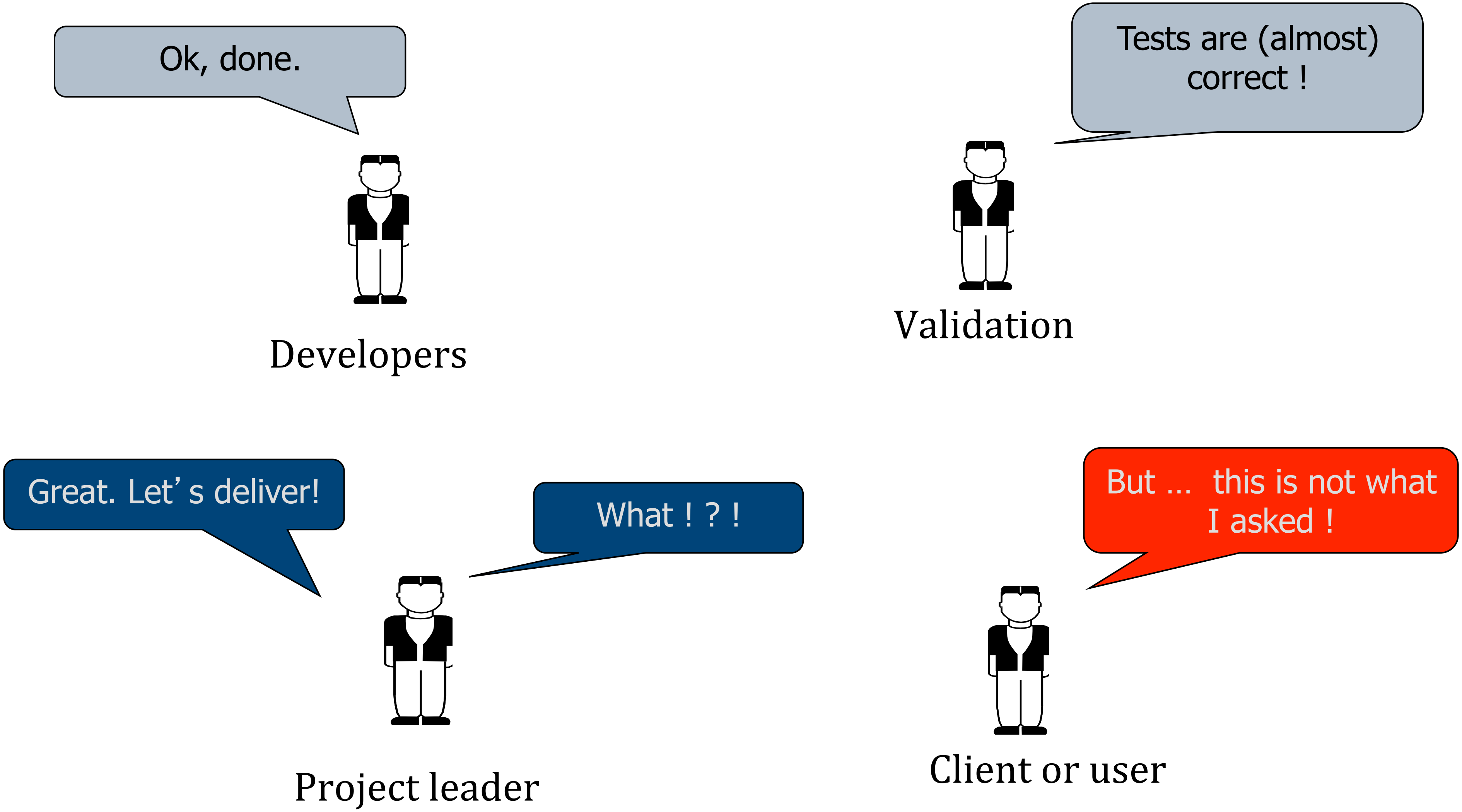
Damn, change your tests!

What ! ? !

Calm down, we are a team !
Where are the **specifications**?

Project leader

The same old story



Ok, done.

Developers

Tests are (almost) correct !

Validation

Great. Let’ s deliver!

What ! ? !

Project leader

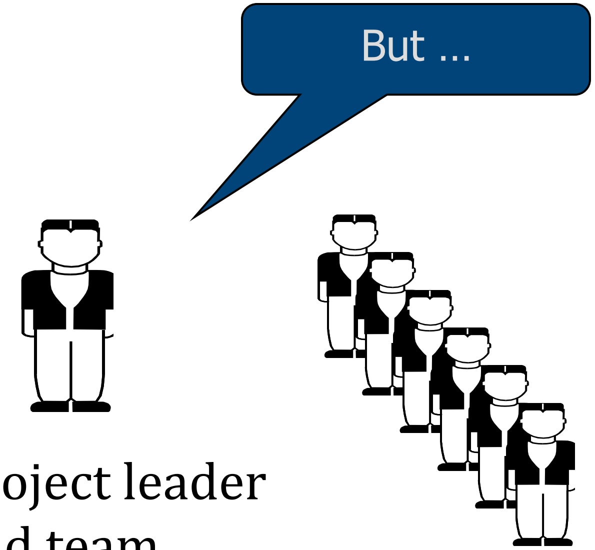
But … this is not what I asked !

Client or user

The same old story

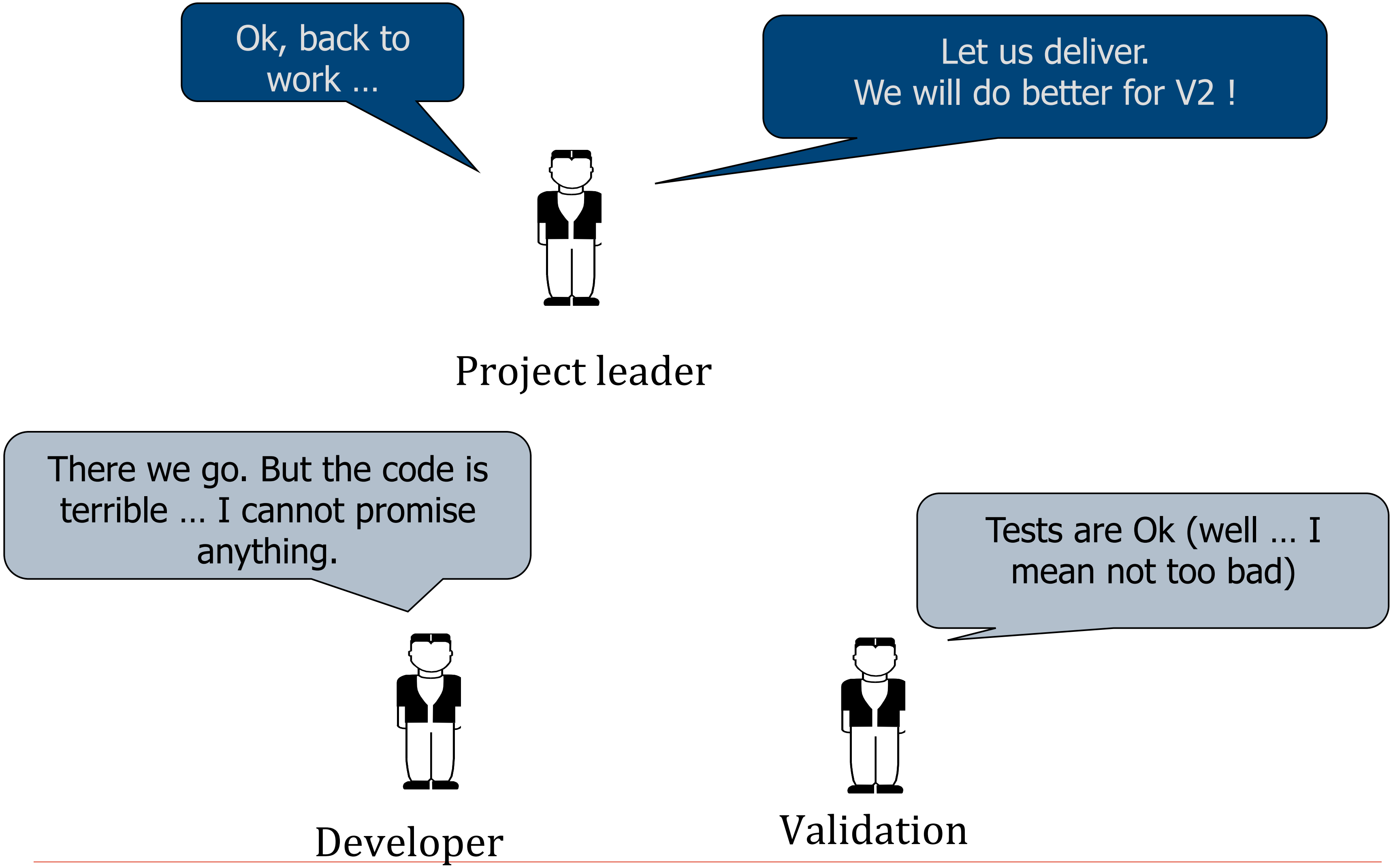
YOU MUST BE KIDDING !!

Manager / Commercial



Project leader
and team

The same old story



Ok, back to work …

Let us deliver.
We will do better for V2 !

Project leader

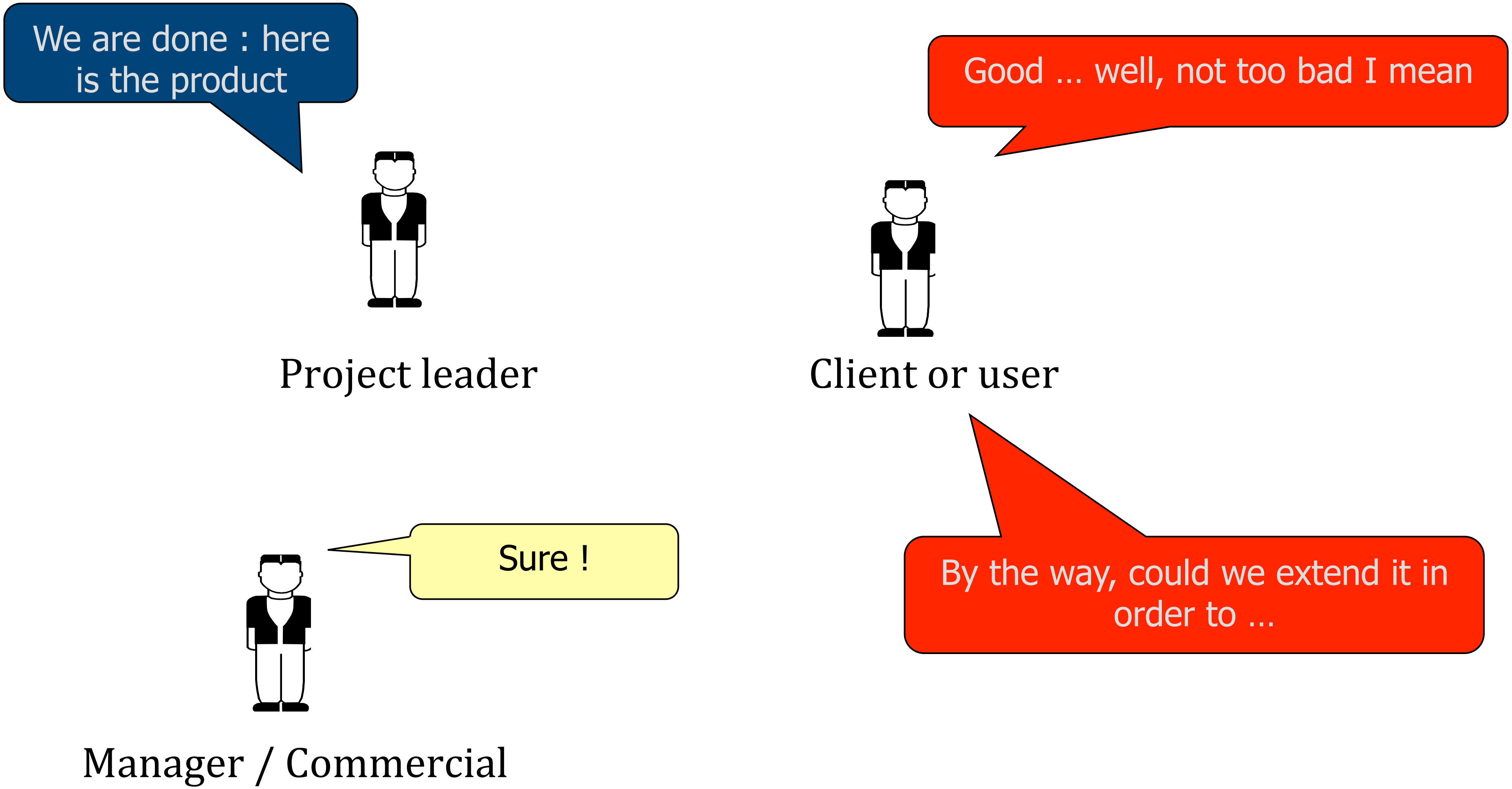
There we go. But the code is terrible … I cannot promise anything.

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 … well, not too bad I mean

Client or user

By the way, could we extend it in order to …

Sure !

Manager / Commercial

Main issues

- This illustrates many issues of software projects
  - Unclear requirements
  - Lack of specification, test preparation, …
  - Bad estimates
  - Lack of support from hierarchy
  - Lack of experience from project leader
  - Lack of users involvement
  - Human problems: communication, skills, …

⇒Purpose of Software Engineering

Purpose of this class - 1

- An introduction to software Engineering
  - Breadth First
- Not a “usual” course
  - Not about a given technology
    - A language, a database system, …
  - Overall understanding sought

Purpose of this class - 2

- At the end, you will understand weird things such as
  - requirements
  - design
  - Architecture
  - Patterns
  - Lifecycle
  - Process
  - Etc.

- And you will know how they relate to each other

Outline

- A short, compelling story
- Historical perspective
- Why software is hard
- Software expected qualities
- Software engineering
- Software engineering activities
- Conclusion

The sixties

- Hardware evolution, first languages
- Emergence of a new job: programmer
  - Distinction between users and programmers
  - 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 OS-360 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

Emerging (insufficient) 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, …

Lessons learned

- There is no single winning solution!
- It has been acknowledged that developing software was an engineering practice
  - Software is an engineering practice <u>in its own</u>
  - Adapted methods, techniques, … are needed
- Software Engineering was born !
  - 1968 (NATO conference)

Eighties and nineties

- Significant improvements in many dimensions
  - Better, more adapted organizations
  - Monitored process, permanent improvement
  - Clear separation: specification / models / programming
  - Structured programming
  - Users involvement
- But the place of software expended notably
  - Projects got bigger and bigger
  - Software systems pervaded many domains
- 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)
  - New driver licenses management (California DMV)
    - stopped after 6 years and $45 spent (' 93)
  - Car&hotel reservation (American Airline, Budget, Hilton)
    - stopped after $165 spent (' 94)
  - Interruption of French train reservation (' 94)
  - Ariane (' 95)

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

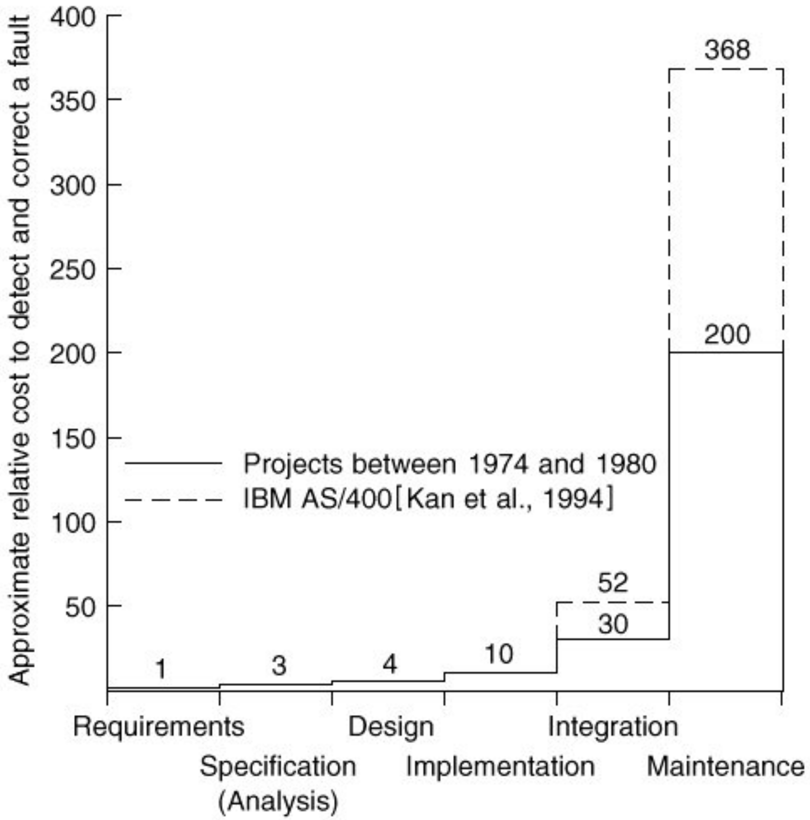
2000’s

- 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

- Bugs and failures represent a true prejudice for companies
  - Financial lost when a project is cancelled
    - Already spent money is lost
  - Missed opportunities
  - Missed commercial launches
  - Lack of desired or necessary functions (in existing soft.)
    - Lack of productivity
    - Lack of services
    - Lack of support
  - Cost of debugging (corrective maintenance)

Cost of finding flaws late

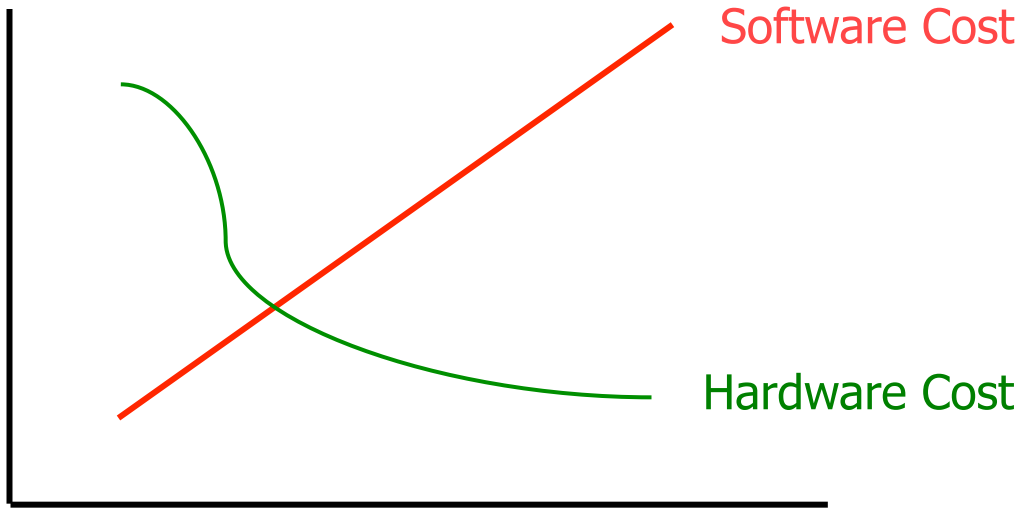


WCB/McGraw-Hill

Outline

- A short, compelling story
- Historical perspective
- Why software is hard
- Software expected qualities
- Software engineering
- Software engineering activities
- Conclusion

Hardware vs. software



- Hardware is getting cheaper and more powerfull
- Why not software???

Why is software so hard?

- We do the hard stuff in software
  - When task is simple & understood, encode it in hardware!
- We have also made progress in SE but
  - Software is everywhere
  - Features explosion
- Size is still growing
- Software is intrinsically hard to build and maintain
- Rising expectations in term of quality (high reliability for instance)

Software is everywhere

OFFICE

RESTAURANT

CAR

HOME

AIRPORT

HOSPITAL

CLUB

STATION

Software size continues to grow

Size	Duration	Programmers	LOC	Examples
Very small	4 months	1	2000	Course project
Small	2 years	3	50K	Pace maker
Medium	3 years	10	100K	Optimizing compiler
Large	5 years	100	1M	MS Word, Excel
Very large	10 years	1000	10M	Air traffic control Space shuttle
Very very large	15+ years	1000	35M	W2K
Ultra large	?	?	?	Pervasive computing Connected health

Software is tough business

- Software is intrinsically hard to produce and maintain for a number of reasons

- 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
  - Not a labor industry!
  - No silver bullet

Software is malleable

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

Software is intangible

- Software is abstract and intangible : it cannot be properly/completely viewed

- Related issues
  - Hard to understand
  - Hard to communicate
  - Hard to manage
    - Time needed for a development?
    - Quality of the resulted software?
    - …

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

- It implies social interactions
  - Communication
  - Management
  - Psychology

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

- Consequences
  - Hard to impose efficient quality politics
  - Hard to convince developers and managers

Conclusion

- Software has to keep up with increasing demands
- Software size continues to grow
- Software has unique characteristics
  - Not to be compared with other domains
  - Not to be ignored
    - “anybody can program …”

- Specific techniques must be defined and used to deal with software projects
  - Software engineering is a discipline

Outline

- A short, compelling story
- Historical perspective
- Why software is hard
- Software expected qualities
- Software engineering
- Software engineering activities
- Conclusion

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 - external
  - Established w.r.t. the requirements specification

- Reliability – external
  - probability that software will operate as expected over a given period of time

Some software qualities (2)

- Robustness – external
  - “reasonable” behavior in unforeseen circumstances
  - a specified requirement is an issue of correctness
  - an unspecified requirement is an issue of robustness

- Usability– external
  - ability of end-users to easily use software
  - extremely subjective

Some software qualities (3)

- Performance – external
  - Ability to meet users expectations in term of rapidity
  - Can be assess by measurement, analysis, and simulation
- Efficiency– internal
  - Usage of internal resources

Some software qualities (3)

- Understandability – internal
  - ability of developers to easily understand produced artifacts
  - Very subjective

- Verifiability – internal
  - ease of establishing desired properties
  - performed by formal analysis or testing

Some software qualities (5)

- Reusability – internal
  - ability to construct new software from existing pieces
  - must be planned for
  - occurs at all levels: requirements, code, processes

- Interoperability – internal
  - ability of software (sub)systems to cooperate with others (for integration purposes)
  - common techniques include APIs, plug-in protocols, etc.

Some software qualities (6)

- Evolvability – internal
  - ability to add or modify functionality (for adaptive and perfective maintenance)
  - evolution should start at requirements or design
- Portability – internal
  - ability to execute in new environments with minimal effort
  - may be planned for by isolating environment-dependent components

Assessing software qualities

- Qualities must be measurable
  - For assessment
  - For improvement
- Measurement requires that qualities be precisely defined
  - Even if incomplete …
- Currently most qualities are informally defined and are then difficult to assess

Outline

- A short, compelling story
- Historical perspective
- Why software is hard
- Software expected qualities
- Software engineering
- Software engineering activities
- Conclusion

Software engineering

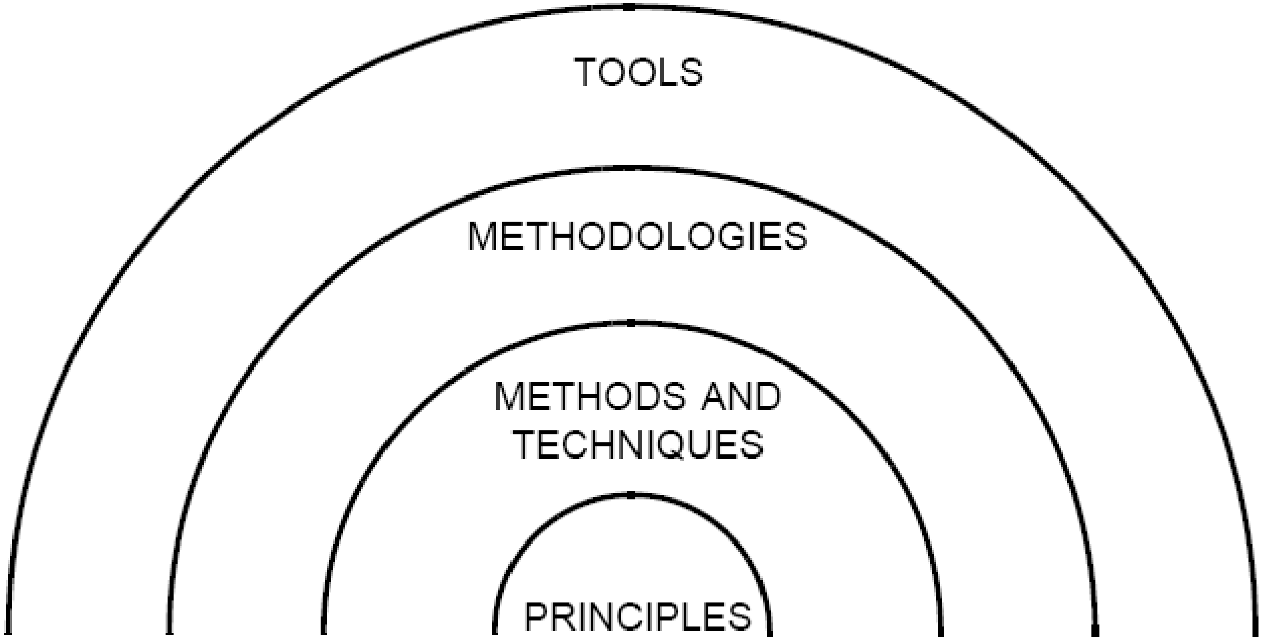
- The study and use of systematic processes and technologies for supporting software development and maintenance activities
  - To control costs
  - To be timely
  - To ensure quality

- Concerned with all aspects of software production
  - From early specifications to late maintenance

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 fields of investigation

- Software processes
  - Definition of activities, roles, …
- Programming languages
  - Structured, object, functional, …
- Methodologies
  - Functional, object (UML), …
- Software management
  - Planning, risk management, outsourcing management
  - Human management
- …

Side note

Software Engineering is not UML!!!!!

Outline

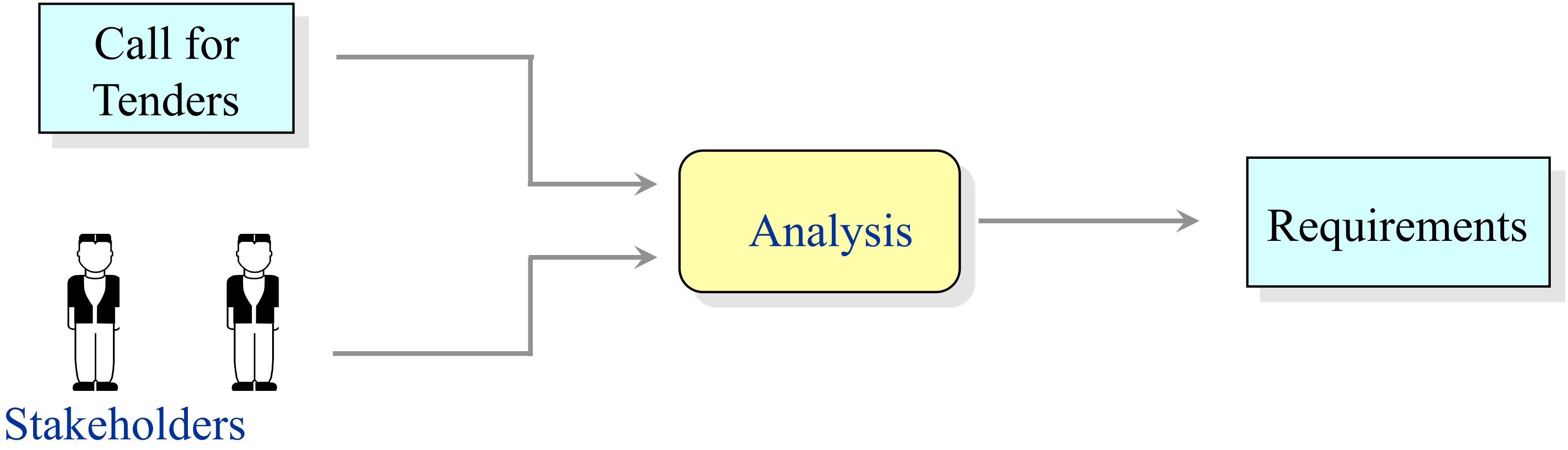
- A short, compelling story
- Historical perspective
- Why software is hard
- Software expected qualities
- Software engineering
- Software engineering activities
- Conclusion

Focus on processes

- Activities
  - Requirements
  - Design
  - Implementation
  - Deployment
  - Maintenance
- Umbrella activities
  - Project management
  - Risk management, etc.

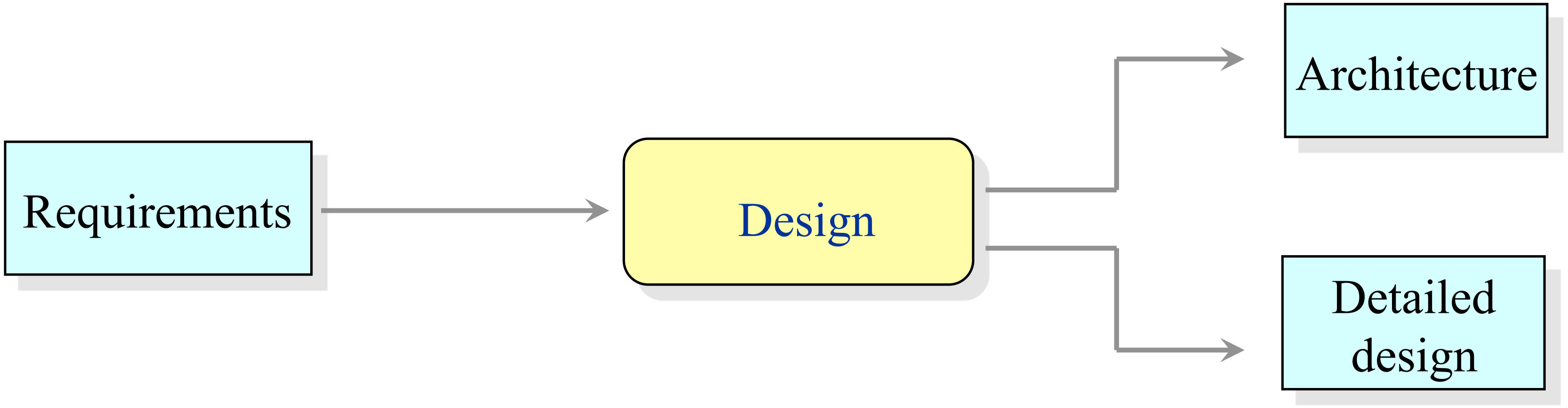
Requirements

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



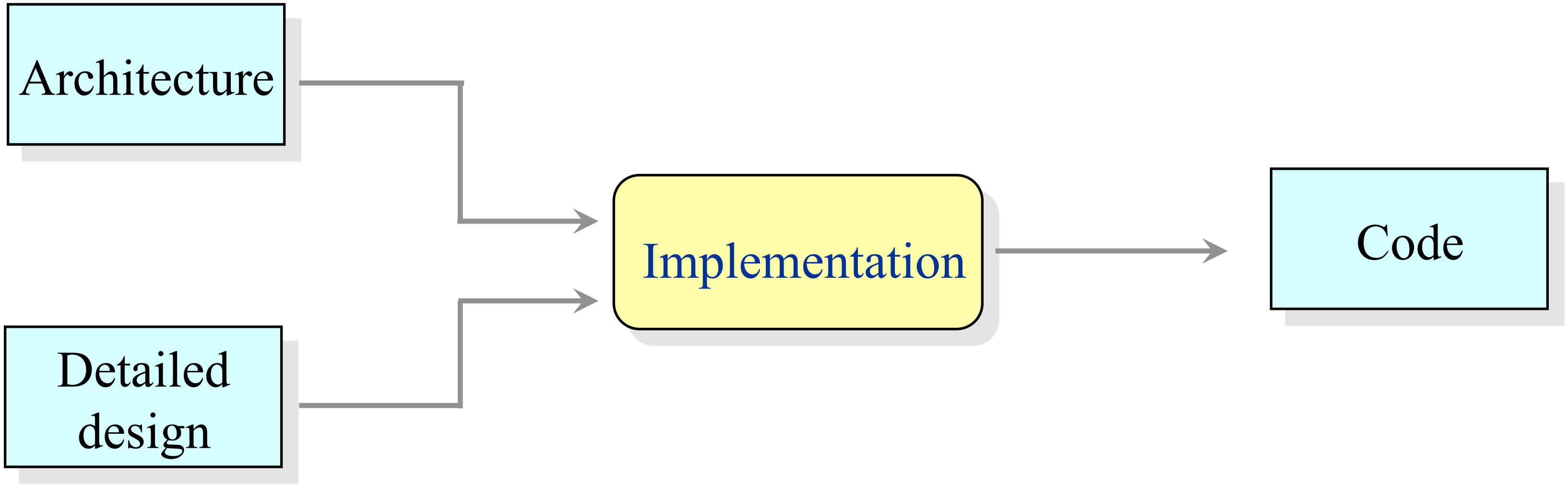
Design

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



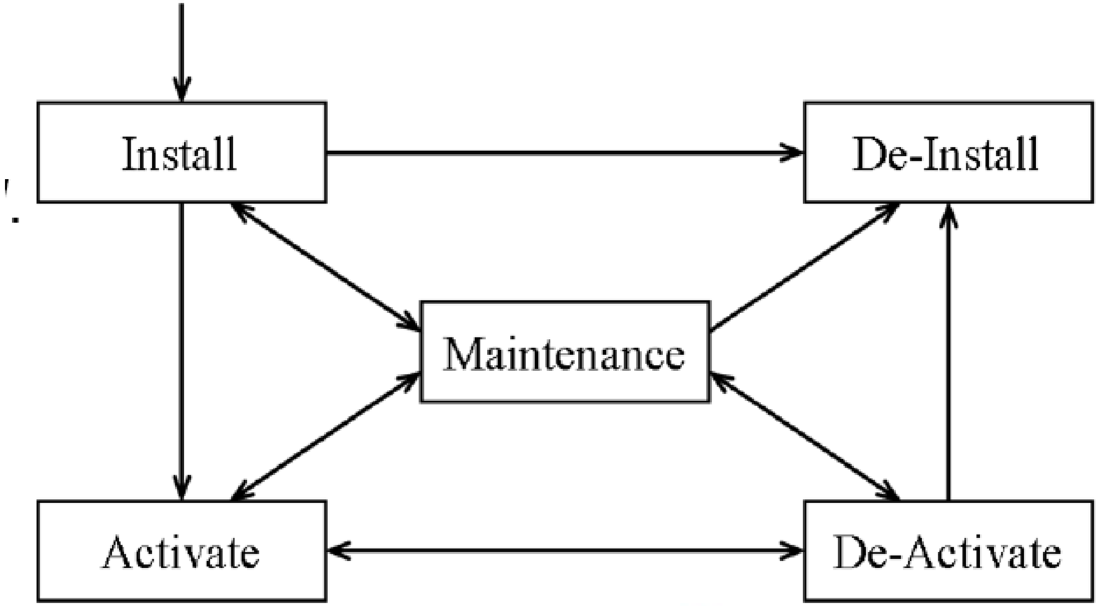
Implementation

- Objectives
  - Create the code meeting the spec



Deployment

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

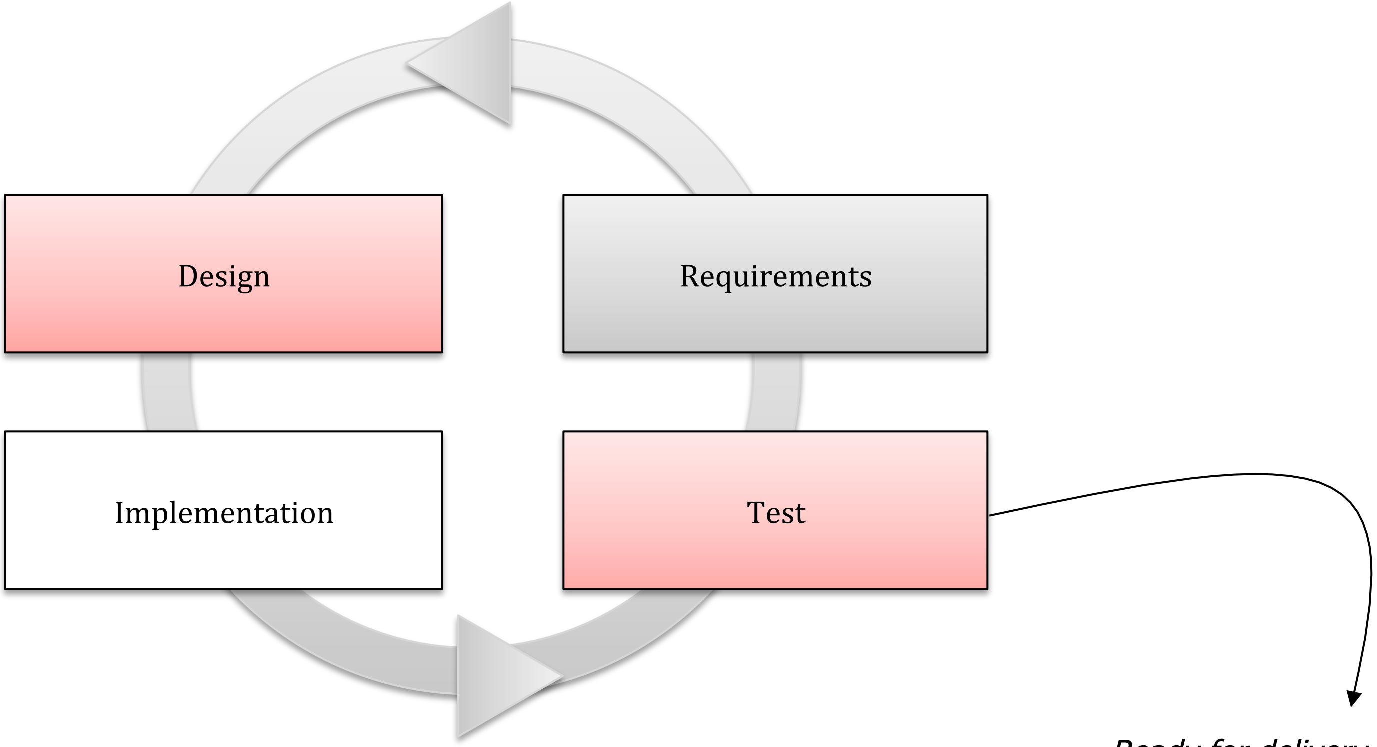


Maintenance

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

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

The way it goes



Design

Requirements

Implementation

Test

Ready for delivery

Outline

□ A short, compelling story

□ Historical perspective

□ Why software is hard

□ Software expected qualities

□ Software engineering

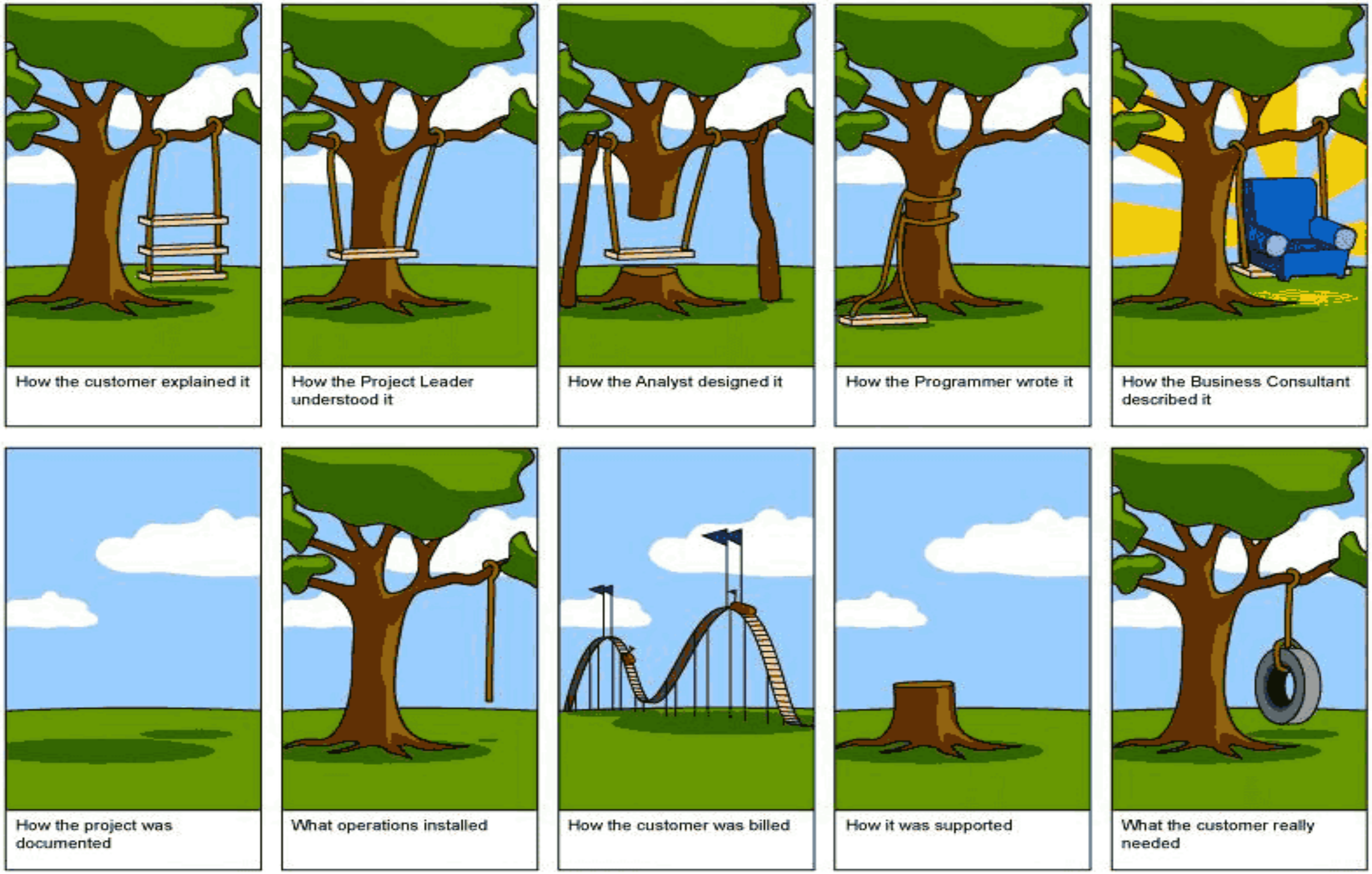
□ Software engineering activities

□ 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