Software Engineering
Spring 2008

Michel Chaudron

Ariadi Nugroho

Outline

- **Introduction**
- **Course logistics**
- **Introductory lecture Software Engineering**
  - What is SE?
  - What does a SE do?
  - What does a SE process look like?

Introduction

Michel Chaudron

- Associate Professor in Leiden (1d) & Eindhoven (4d)
- Ph.D. students: Ariadi Nugroho (assistant) & Werner Heijstek

- M.Sc. & Ph.D. from Leiden, some time abroad
- some years with IT company
- research in software engineering:
  - software architecture and component-based sw engineering
  - quality, measurement in SE – esp. UML
- Collaborations with companies: Philips, Oce, CapGemini, LogicaCMG, KLM, Nokia, …

What you will learn?

Engineering = skill + knowledge

This course 80% knowledge and 20% skills

Basic concepts, vocabulary of Software Engineering

Main activities in SE projects

Main methods and techniques (excluding: programming)

Guest Lectures by professionals

SE as an academic research area

Book: Object-Oriented Software Engineering, Timothy C. Lethbridge, Robert Laganière (2nd Ed.)

Ch 1: introduction to the subject

Ch 2: OO-basics

Ch 4: Requirements

Ch 5 & Ch 8: Modeling using UML

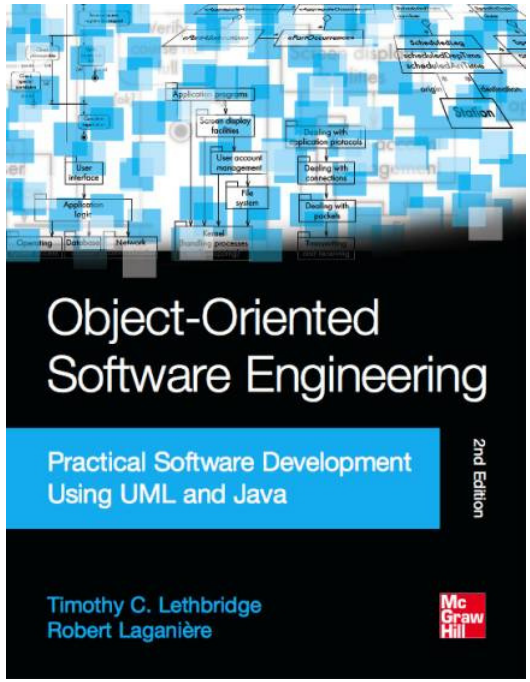
Ch 6: Design patterns

Ch 9: Architecture & Designing

Ch 10: Testing / Quality Assurance

Ch 11: Management (Estimation, Risk)

Websites: www.mhhe.com/lethbridge en www.llsoeng.com



Assignment

Car Navigation System

- Requirements
- Architecture & Design
- Analysis
- Implementation (mock-up)
- Test

Lectures Schedule

Wk-nr		Datum	lecturer	onderwerp	Huiswerk/leeswerk
6	1	7 feb	Chaudron	Introduction Software Engineering	LL Ch 1, 2
7	2	14 feb	Chaudron	Requirements Engineering	LL Ch.4.
8	3	21 feb	Chaudron	Software Architecting	LL Ch. 9
9	4	28 feb	Chaudron	Modeling with UML	LL Ch 5
10	5	6 maart	Peter Bink	Cost Estimation, Planning & Control	LL Ch 11
11	6	13 maart	Chaudron	Software Metrics	LL Ch 10 & 11
12	7	20 maart	Bart Kienhuis	Design Patterns / Refactoring	LL Ch 6
13	8	27 maart	Chaudron	onderzoeksmethoden empirisch onderzoek in software engineering	
14	9	3 april	Bart Knaack	Testing & Quality Assurance (Requirements, Design, Code)	LL Ch. 10
15	10	10 april	Rijn Buve?	Gastspreker (KLM ? / TomTom?)	
16	11	17 april	Rijn Buve?		
17	12	24 april			
18		1 mei	--	Hemelvaart	
19	13	8 mei	Chaudron	Vragen-uur	

Object-Oriented Software Engineering
Practical Software Development using UML and Java

Chapter 1: Software and Software Engineering

What is Software Engineering?

What is SW quality?

What is a software development process?

1.1 The Nature of Software...

Software is intangible

- Hard to understand development effort

Software is easy to reproduce

- Cost is in its *development*

  —in other engineering products, manufacturing is the costly stage

The industry is labor-intensive

- Hard to automate

The Nature of Software ...

Untrained people can hack something together

- Quality problems are hard to notice

Software is easy to modify

- People make changes without fully understanding it

Software does not 'wear out'

- It deteriorates by having its design changed:
  - —erroneously, or
  - —in ways that were not anticipated, thus making it complex

The Nature of Software

Conclusions

- Much software has poor design and is getting worse
- Demand for software is high and rising
- We are in a perpetual ‘software crisis’
- We have to learn to ‘engineer’ software

Types of Software...

Custom

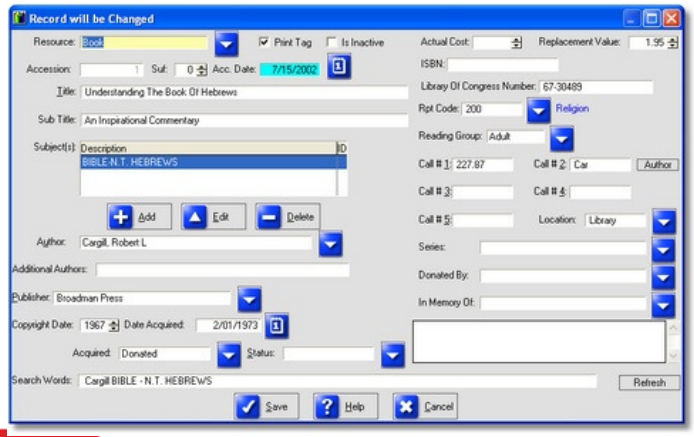
- For a specific customer

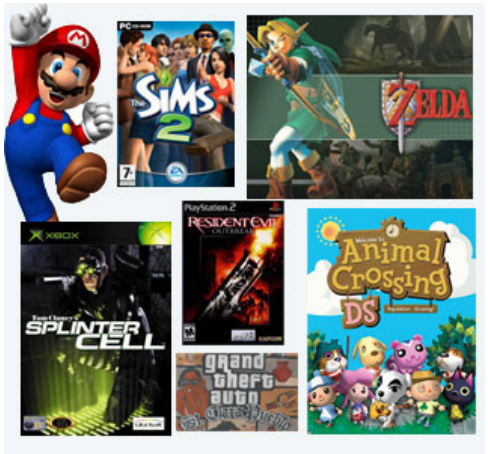
Generic

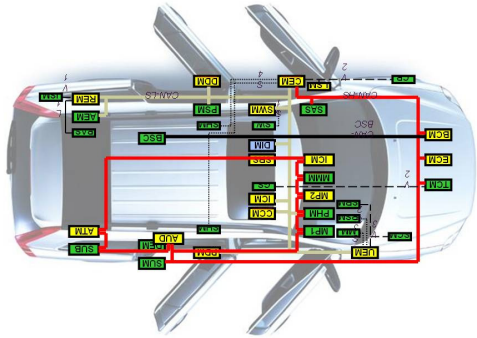
- Sold on open market
- Often called
  - —COTS (Commercial Off The Shelf)
  - —Shrink-wrapped

Embedded

- Built into hardware
- Hard to change

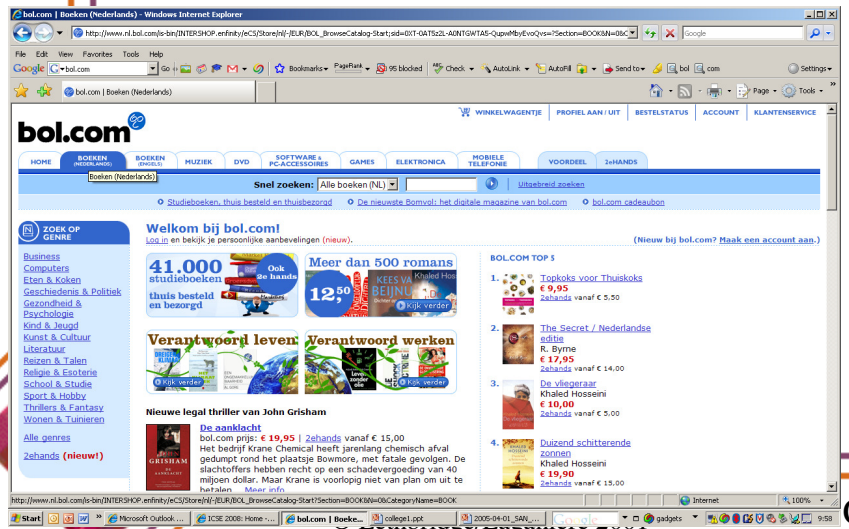


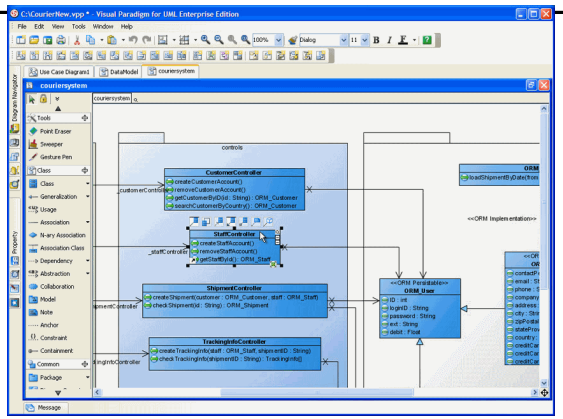




Types of Software

	Custom	Generic	Embedded
Number of *copies* in use	low	medium	high
Total *processing power* devoted to running this type of software	low	high	medium
Worldwide annual *development effort*	high	low	high





Types of Software

Real time software

- E.g. control and monitoring systems
- Must react immediately
- Safety often a concern

Business Information Systems (Data processing)

- Used to run businesses
- Accuracy and security of data are key

Some software has both aspects

1.2 What is Software Engineering?...

The process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints

Solving customers' problems

- This is the goal of software engineering
- Sometimes the solution is to buy, not build
- Adding unnecessary features does not help solve the problem
- Software engineers must communicate effectively to identify and understand the problem

What is Software Engineering?…

Systematic development and evolution

- An engineering process involves applying well understood techniques in a organized and disciplined way
- Many well-accepted practices have been formally standardized
  —e.g. by the IEEE or ISO

Large, high quality software systems

- Software engineering techniques are needed because large systems cannot be completely understood by one person
- Teamwork and co-ordination are required
- Key challenge: Dividing up the work and ensuring that the parts of the system work properly together
- The end-product that is produced must be of sufficient quality

What is Software Engineering?...

Other definitions:

- IEEE: (1) the application of a systematic, disciplined, quantifiable approach to the development, operation, maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)

- The Canadian Standards Association: The systematic activities involved in the design, implementation and testing of software to optimize its production and support.

What is Software Engineering?

Cost, time and other constraints

- Finite resources
- The benefit must outweigh the cost
- Others are competing to do the job cheaper and faster
- Inaccurate estimates of cost and time have caused many project failures

What is the Science of Software Engineering?

The scientific study of
methods, techniques, processes
for creating software

Effect of techniques on quality, productivity

Object Oriented programming languages are better.

Agile development processes lead to faster development.

Often studied empirically

1.4 Stakeholders in Software Engineering

1. Users

- Those who use the software

2. Customers

- Those who pay for the software

3. Software developers

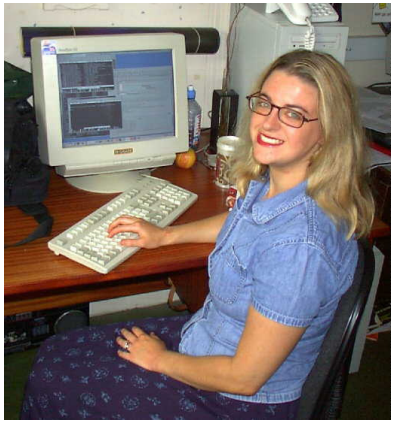
- Those who make the software

4. Development Managers

All four roles can be fulfilled by the same person

What does a Software Engineer do?

individually



programming
documenting
planning
presenting
reviewing
reporting

interacting
with clients

listening
explaining
feedback
selling

in team

Specializing in different roles

- designing, programming, testing …

brainstorming
discussing
planning

1.5 Software Quality...

Usability

- Users can learn it and fast and get their job done easily

Efficiency

- It doesn't waste resources such as CPU time and memory

Reliability

- It does what it is required to do without failing

Maintainability

- It can be easily changed

Reusability

- Its parts can be used in other projects, so reprogramming is not needed

Software Quality...

Customer:
solves problems at
an acceptable cost in
terms of money paid and
resources used

User:
easy to learn;
efficient to use;
helps get work done

Developer:
easy to design;
easy to maintain;
easy to reuse its parts

Development manager:
sells more and
pleases customers
while costing less
to develop and maintain

Software Quality

The different qualities can conflict

- Increasing efficiency can reduce maintainability or reusability
- Increasing usability can reduce efficiency

Setting objectives for quality is a key engineering activity

- You then design to meet the objectives
- Avoids ‘over-engineering’ which wastes money

Optimizing is also sometimes necessary

- E.g. obtain the highest possible reliability using a fixed budget

Internal Quality Criteria

These:

- Characterize aspects of the design of the software
- Have an effect on the external quality attributes
- E.g.
  - —The amount of commenting of the code
  - —The complexity of the code

Short Term Vs. Long Term Quality

Short term:

- Does the software meet the customer's immediate needs?
- Is it sufficiently efficient for the volume of data we have today?

Long term:

- Maintainability
- Customer's future needs

1.6 Software Engineering Projects

Most projects are *evolutionary* or *maintenance* projects, involving work on *legacy* systems

- Corrective projects: fixing defects
- Adaptive projects: changing the system in response to changes in
  - —Operating system
  - —Database
  - —Rules and regulations
- Enhancement projects: adding new features for users
- Reengineering or perfective projects: changing the system internally so it is more maintainable

Software Engineering Projects

‘Green field’ projects

- New development
- The minority of projects

Software Engineering Projects

Projects that involve building on a *framework* or a set of existing components.

- The framework is an application that is missing some important details.
  - —E.g. Specific rules of this organization.
- Such projects:
  - —Involve plugging together *components* that are:
    - Already developed.
    - Provide significant functionality.
  - —Benefit from reusing reliable software.
  - —Provide much of the same freedom to innovate found in green field development.

1.7 Activities Common to Software Projects...

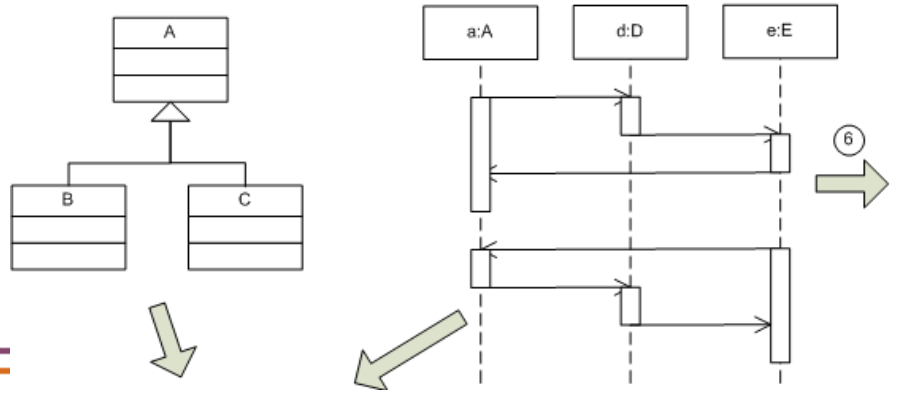
Requirements and specification

- Includes
  - —Domain analysis
  - —Defining the problem
  - —Requirements gathering
    - Obtaining input from as many sources as possible
  - —Requirements analysis
    - Organizing the information
  - —Requirements specification
    - Writing detailed instructions about how the software should behave

Activities Common to Software Projects...

Design

- Deciding how the requirements should be implemented, using the available technology
- Includes:
  - —Systems engineering: Deciding what should be in hardware and what in software
  - —Software architecture: Dividing the system into subsystems and deciding how the subsystems will interact
  - —Detailed design of the internals of a subsystem
  - —User interface design
  - —Design of databases



Activities Common to Software Projects

Modeling

- Creating representations of the domain or the software
  - —Use case modeling
  - —Structural modeling
  - —Dynamic and behavioural modeling

Programming

Quality assurance

- Reviews and inspections
- Testing

Deployment

Managing the process

1.8 The Eight Themes of the Book

1. **Understanding the customer and the user**
2. **Basing development on solid principles and reusable technology**
3. **Object orientation**
4. **Visual modeling using UML**
5. **Evaluation of alternatives**
6. **Iterative development**
7. **Communicating effectively using documentation**
8. **Risk management in all SE activities**

Difficulties and Risks in Software Engineering

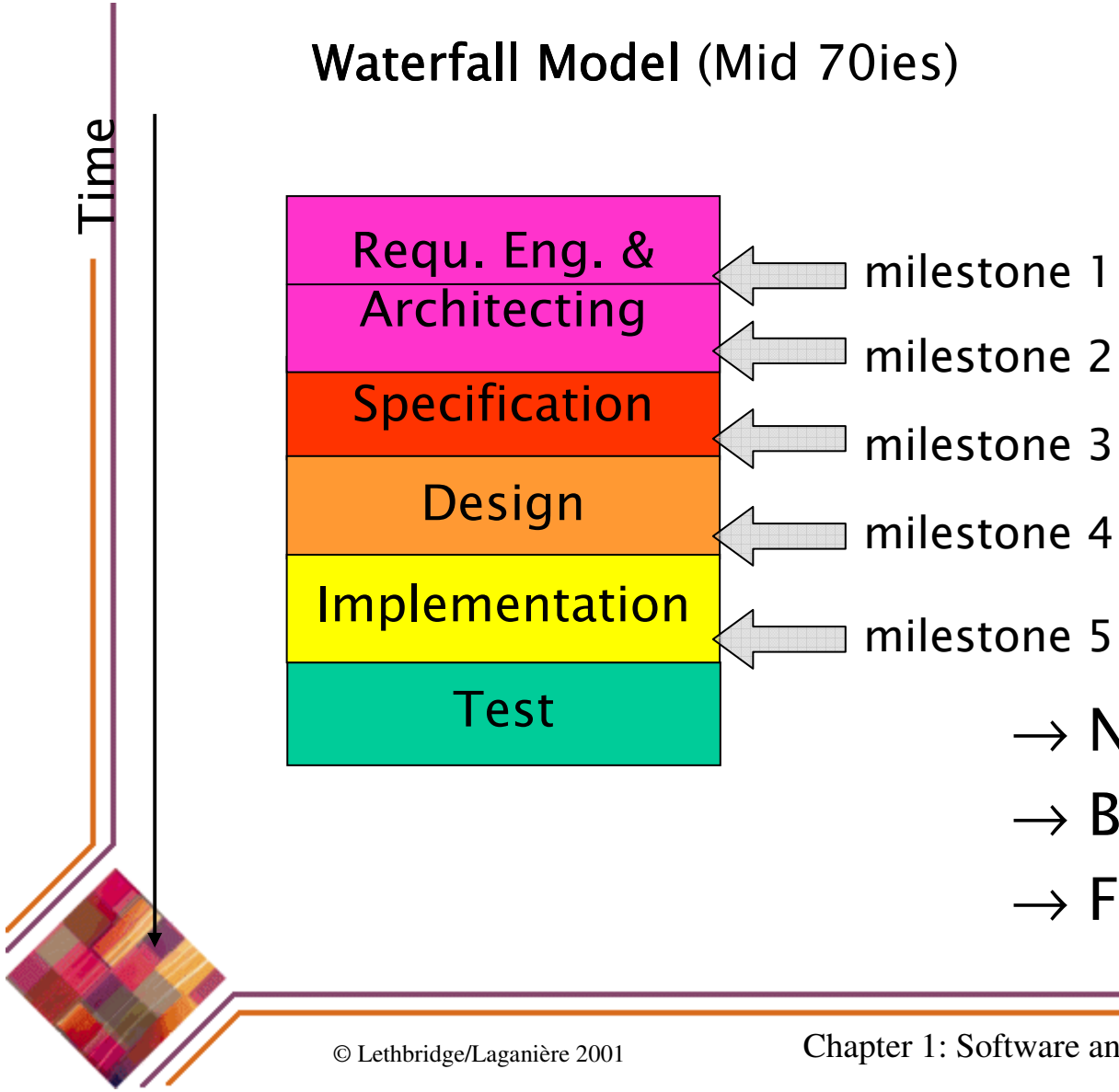
- **Complexity and large numbers of details**
- **Uncertainty about technology**
- **Uncertainty about requirements**
- **Uncertainty about software engineering skills**
- **Constant change**
- **Deterioration of software design**
- **Political risks**

Software Development Process Models

- **Waterfall**
- **Iterative**

SDP Models (1)

Waterfall Model (Mid 70ies)

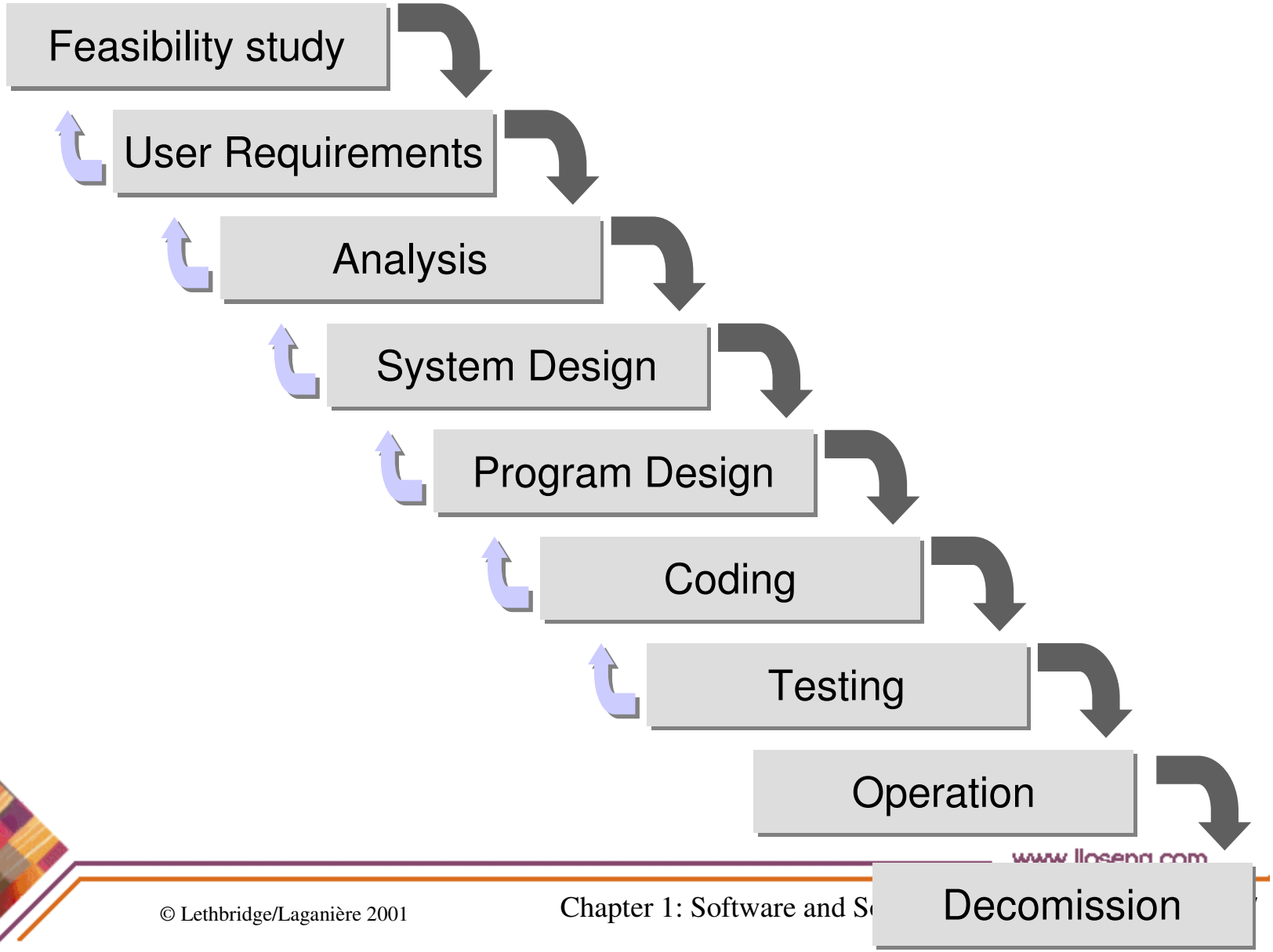


→ No iterations

→ Big bang scenario

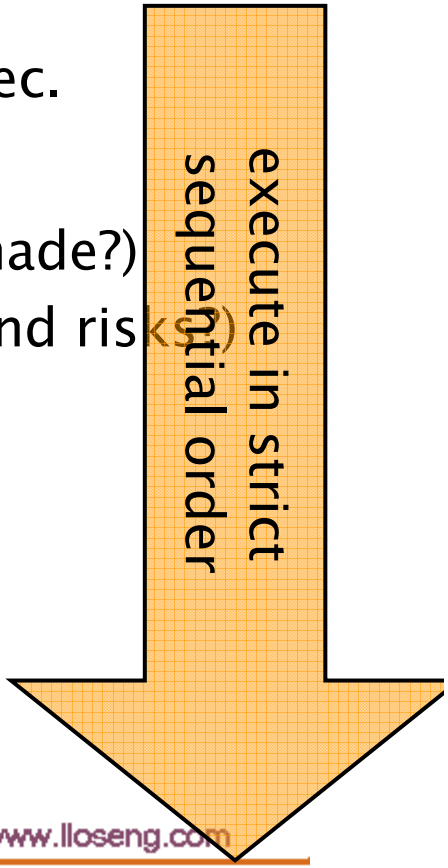
→ First-time right

The waterfall model



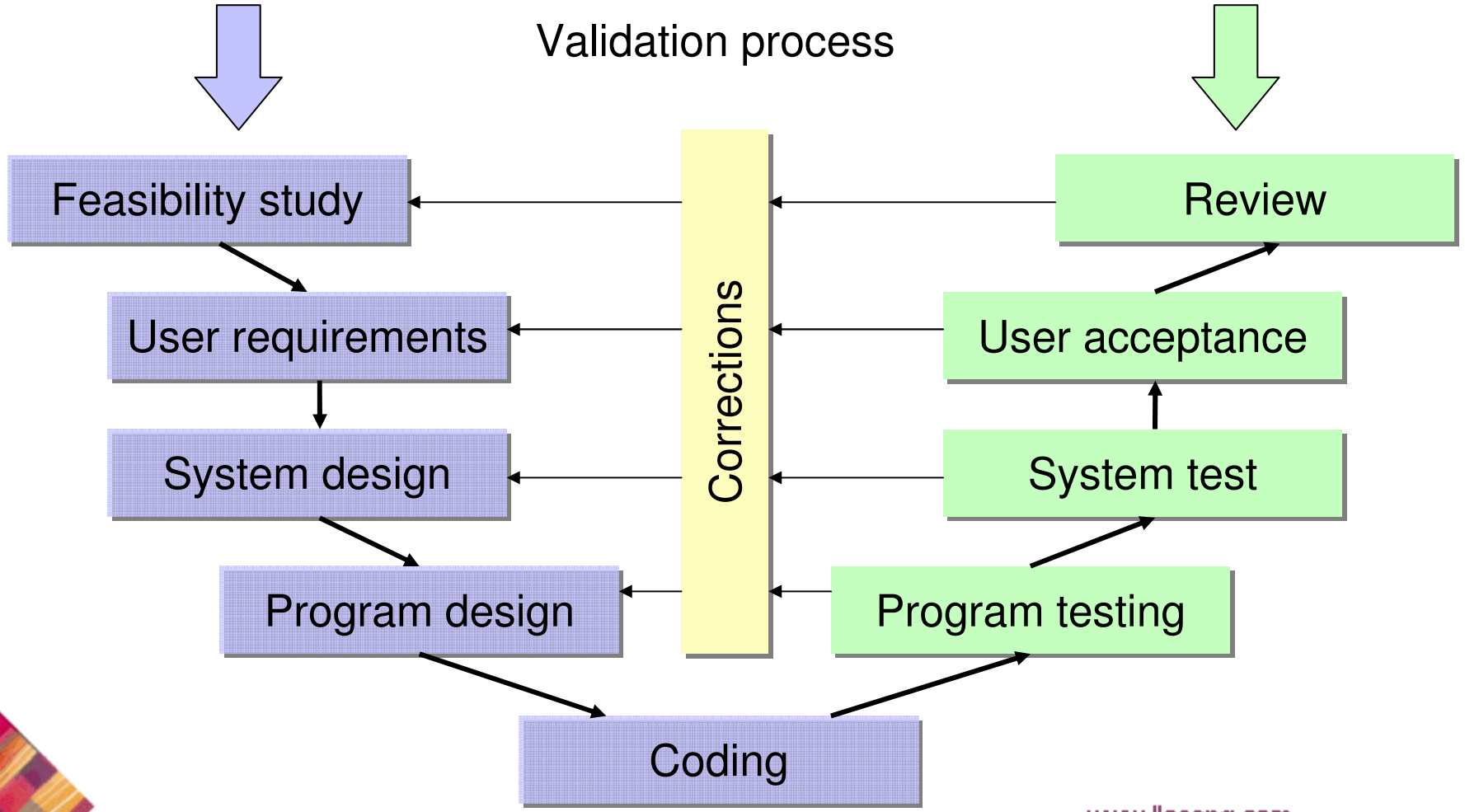
The Classical Waterfall Model (Example)

Requirements	Vision & first idea
Analysis	Requirements Document (WHAT)
	Context model & Requirements Spec.
Architectural Model (HOW)	
	Feasibility Study (can product be made?)
	Risk Assessment (project threats and risks)
Design & Specification	
	System Spec. (WHAT):
	Design (HOW)
Implementation	Coding & Testing (HOW):
Test	Integration and acceptance Test



The V-process model

Another way of looking at the waterfall model



Problems of the Waterfall Process (1)

The milestones did not fit in many project situations, leading to:

- **Gold-plating** → Iterative development

  Extensive written requirements spec's cause overemphasis on "complete" requirements and invite "just-in-case" additions

- **Inflexible point solutions**

  – Fixed requirements spec's produce inflexible solutions optimized around the initial problem statement

  – Forced early design decisions

- **Bad usability** → A prototype is worth a 100.000 words

  Written req. spec's are not nearly as effective as a prototype

  Requirements often emerge only after demonstration and feedback

The waterfall model (cont'd)

Pros:

Imposes structure on complex projects

Every stage needs to be checked and signed off:

- Elimination of midstream changes

Good when quality requirements dominate cost and schedule requirements

Cons:

Limited scope for flexibility / iterations

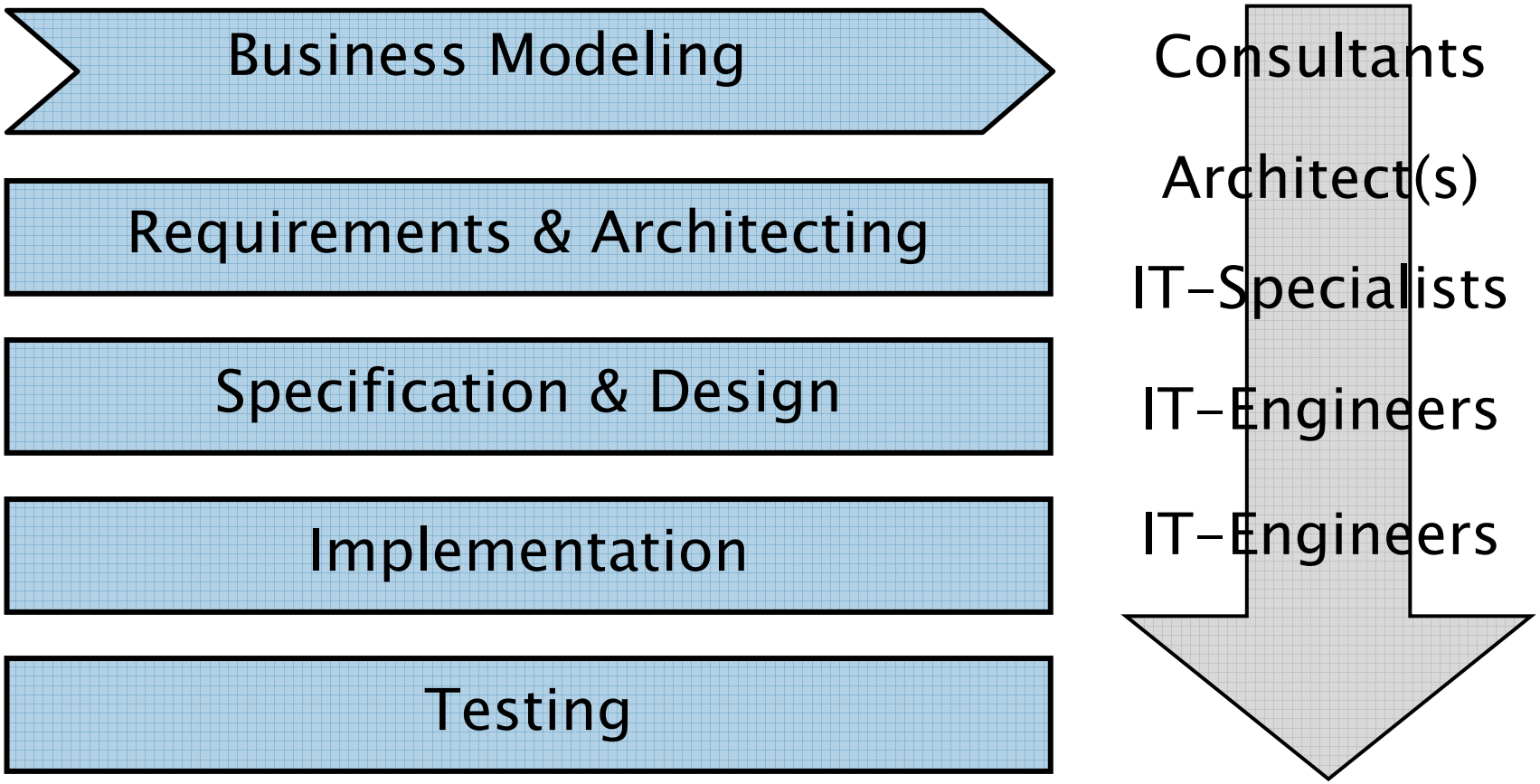
Full requirements specification at the beginning:

- User specifications

No tangible product until the end

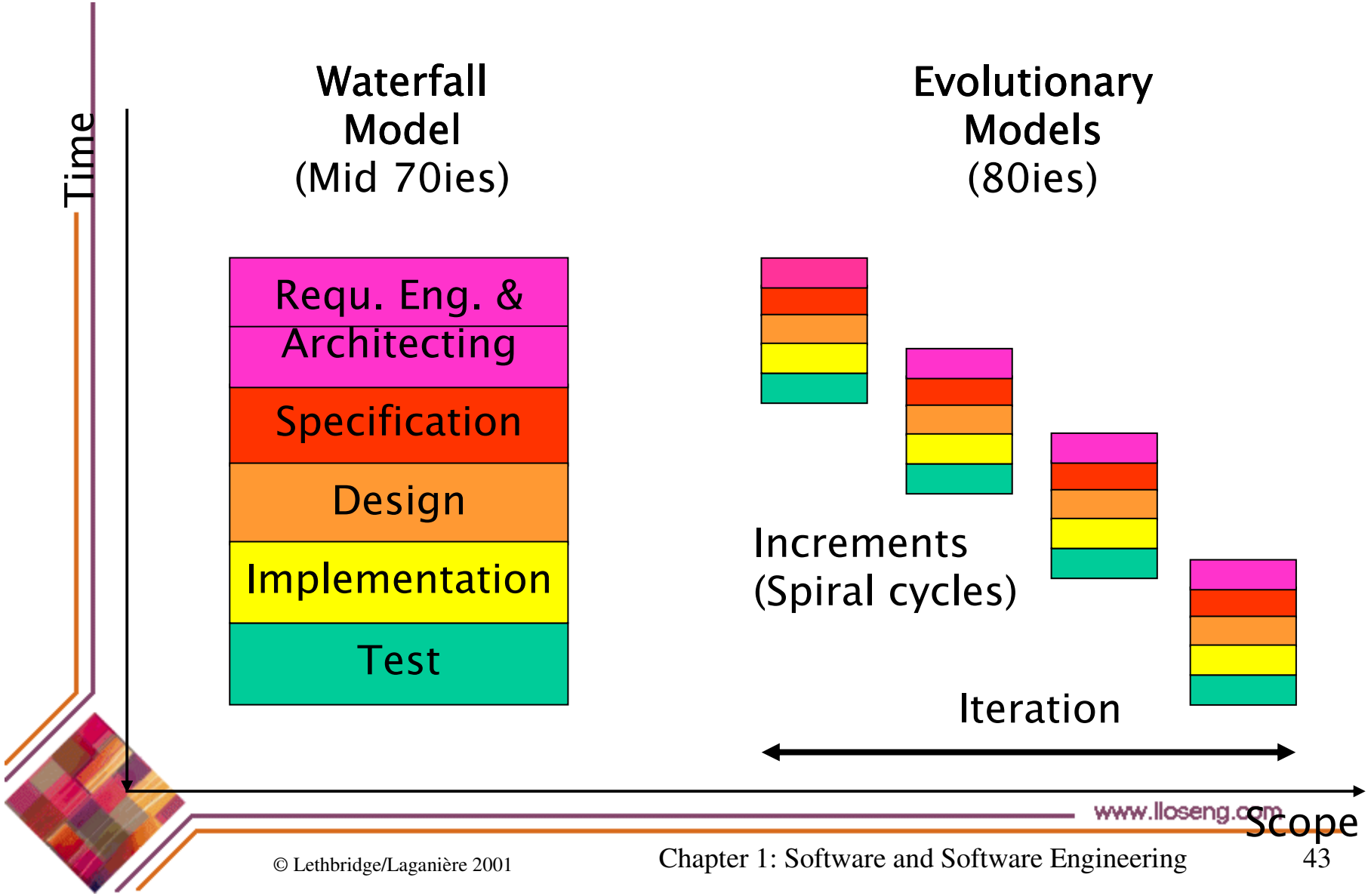
Problems of the Waterfall Process (2)

Different phases are handled by different people

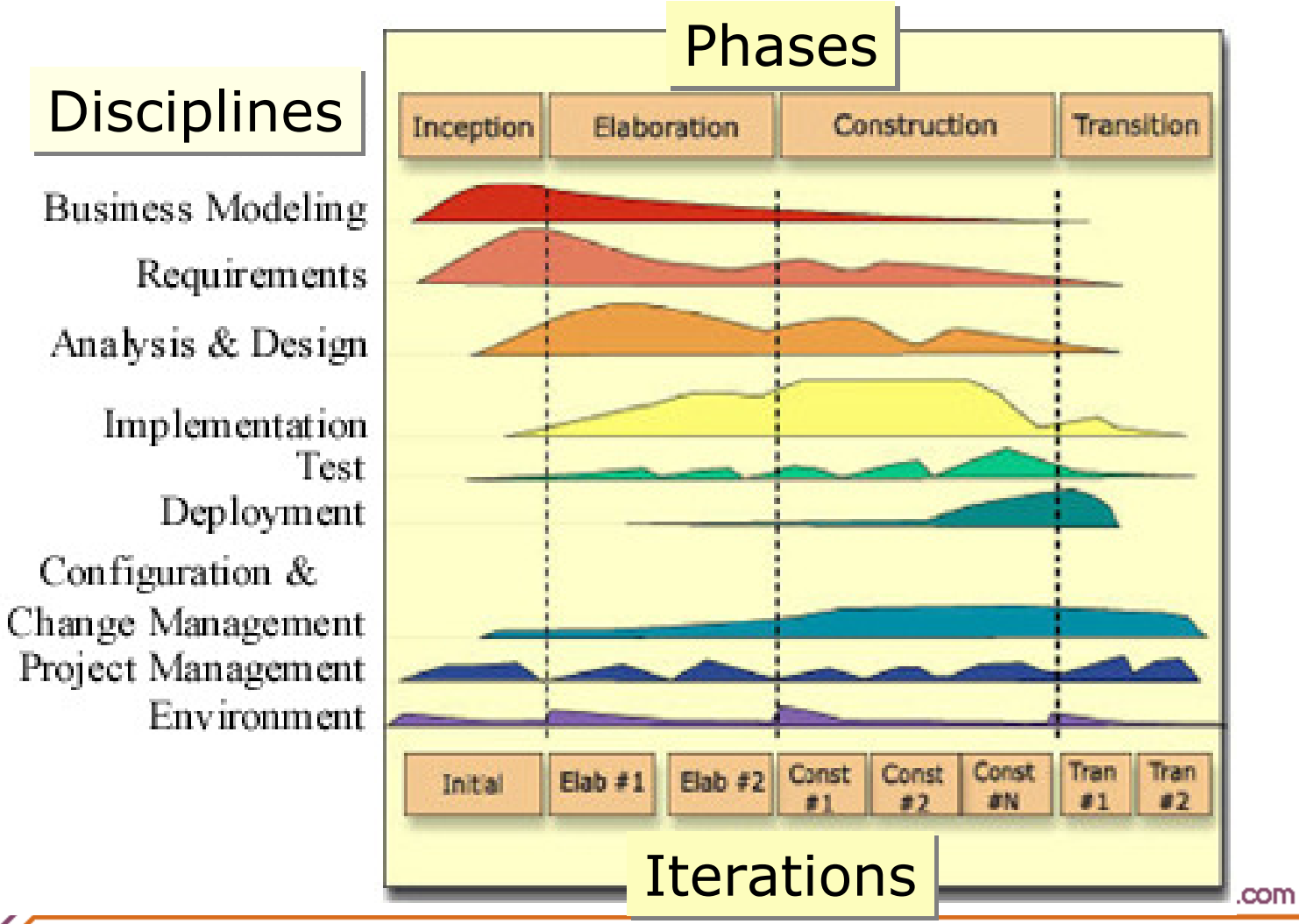


Communication becomes highly critical

SDP Models (2)


Time
Waterfall Model (Mid 70ies)
Requ. Eng. & Architecting
Specification
Design
Implementation
Test
Evolutionary Models (80ies)
Increments (Spiral cycles)
Iteration
www.lloseng.com
Scope

Rational Unified Process (RUP)



Problems of Evolutionary Models

- **Inflexible point solutions**

  The initial release is optimized for demonstration, consequently the architecture is difficult to extend

- **High-risk downstream capabilities**

  The initial release often defers quality attributes (dependability, scalability, etc.) in favor of early functionality

Win-Win Spiral Model

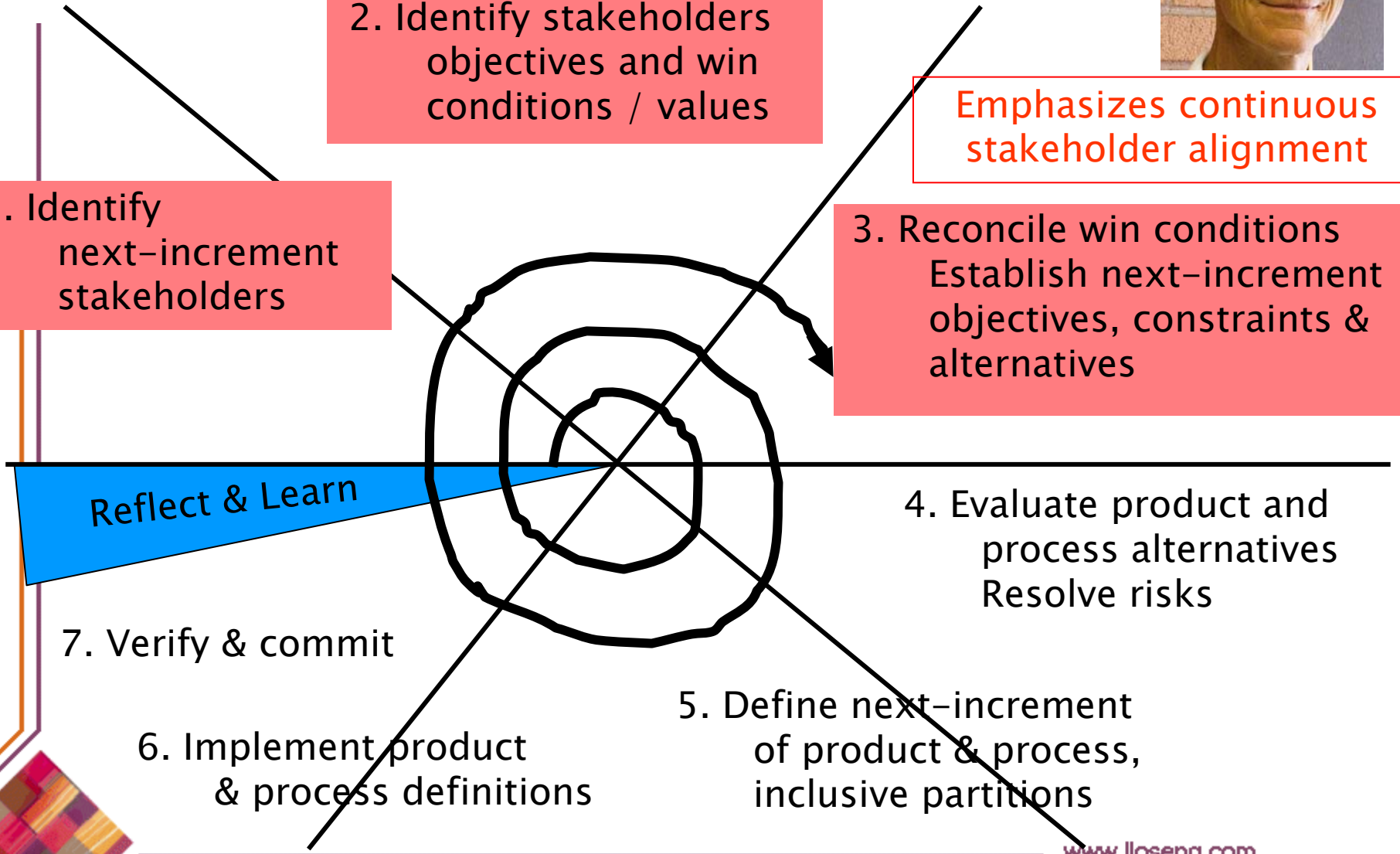
(Boehm, 1998)

2. Identify stakeholders objectives and win conditions / values

Emphasizes continuous stakeholder alignment

1. Identify next-increment stakeholders

3. Reconcile win conditions
Establish next-increment objectives, constraints & alternatives



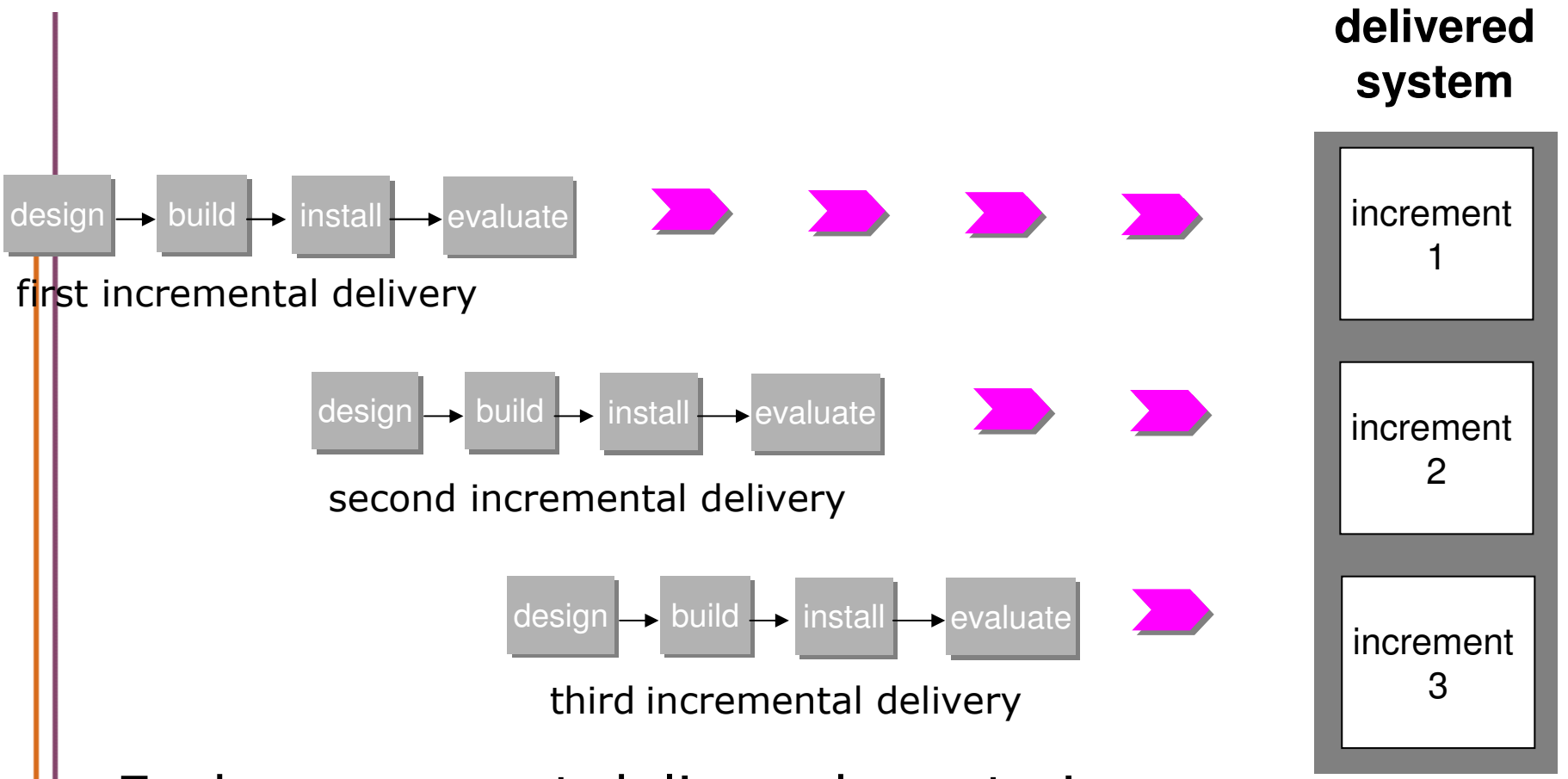
4. Evaluate product and process alternatives
Resolve risks

7. Verify & commit

5. Define next-increment of product & process, inclusive partitions

6. Implement product & process definitions

Incremental delivery



Each component delivered must give some benefit to the stakeholders

Proliferation of Alternative Models

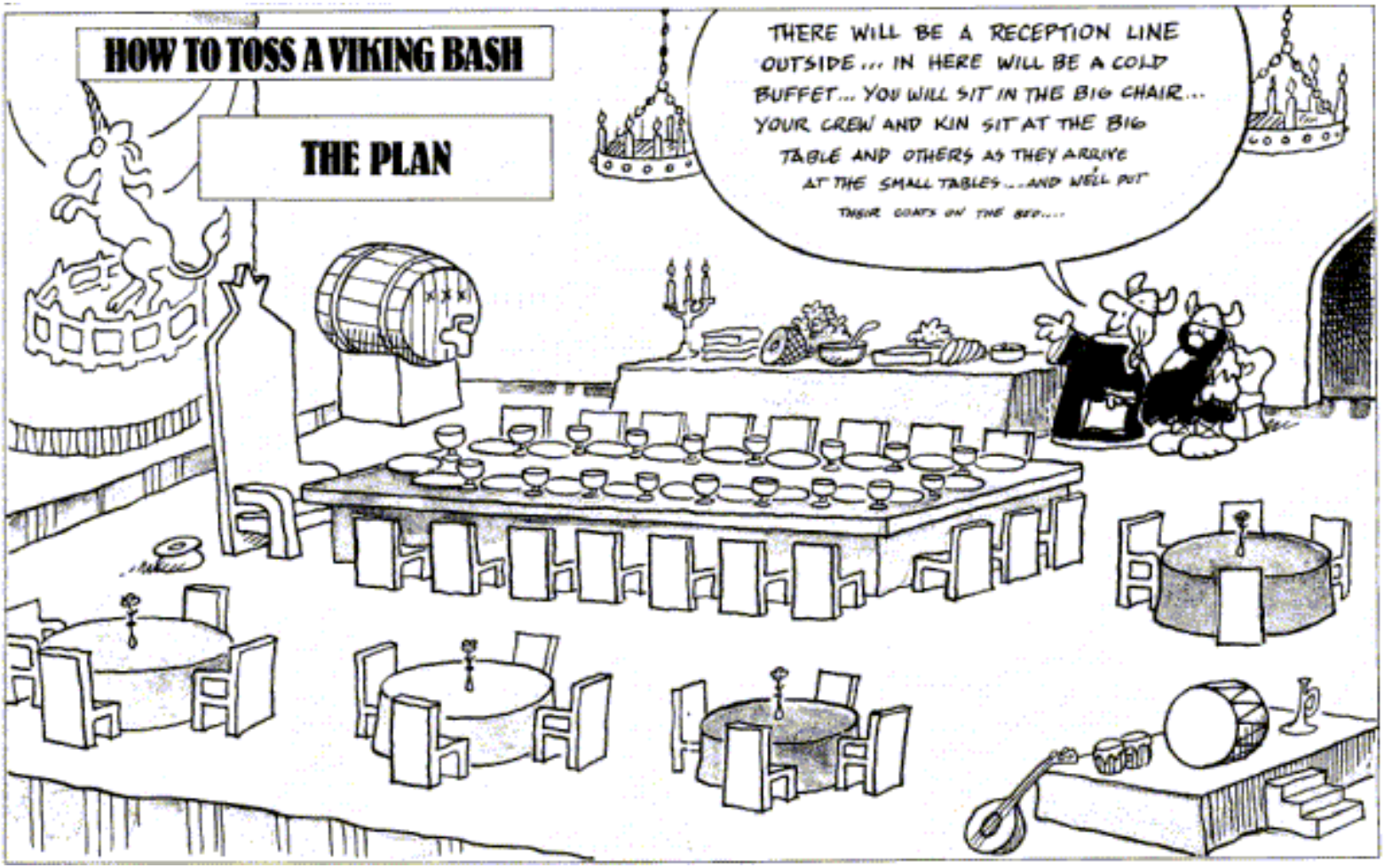
Early 1990's

Examples:

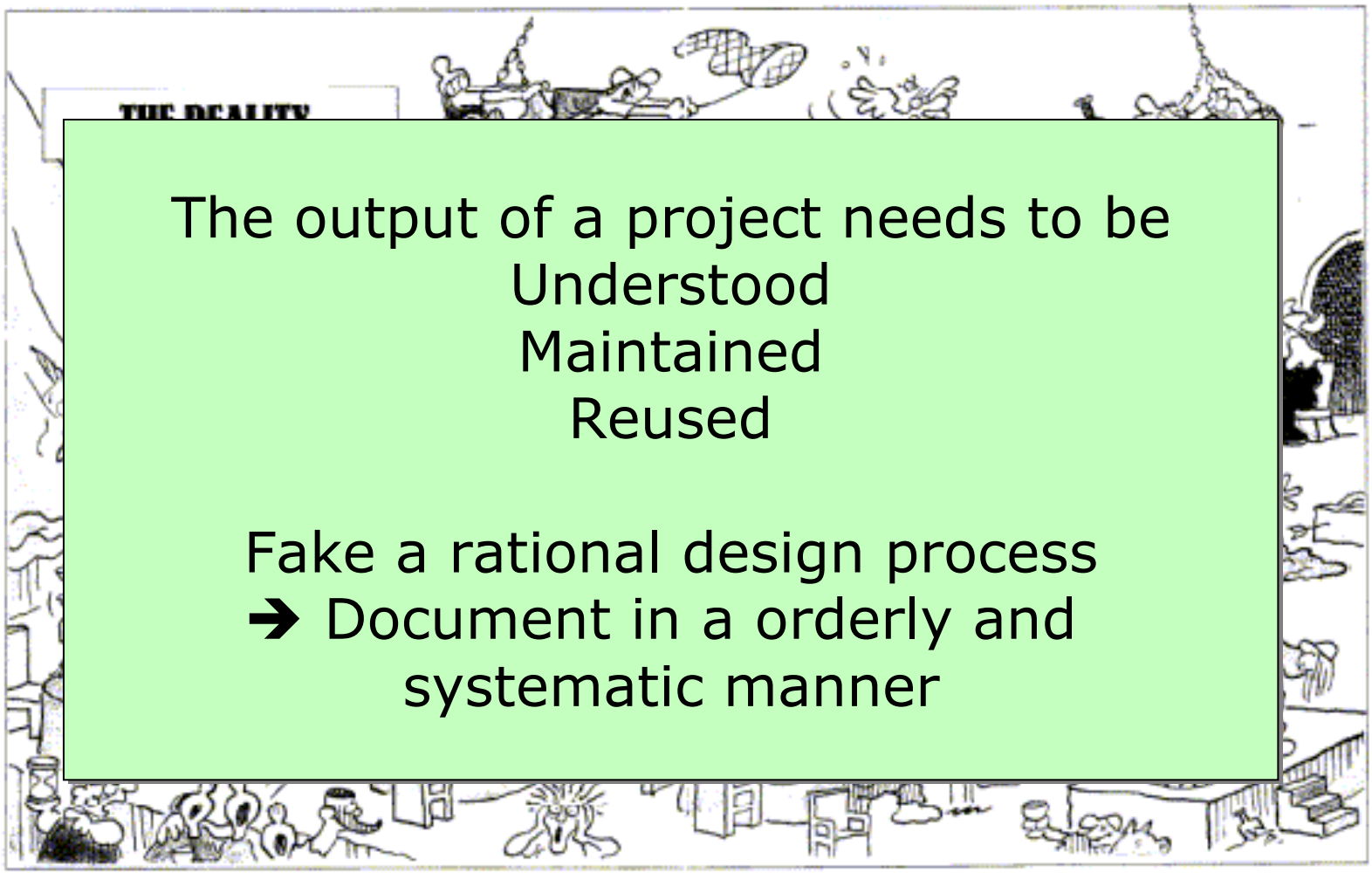
- Risk-, reuse-, legacy- and demo-driven
- Various variants of evolutionary development
- Hybrids

SW organizations had difficulties
to establish a common reference

The plan


HOW TO TOSS A VIKING BASH
THE PLAN
THERE WILL BE A RECEPTION LINE OUTSIDE... IN HERE WILL BE A COLD BUFFET... YOU WILL SIT IN THE BIG CHAIR... YOUR CREW AND KIN SIT AT THE BIG TABLE AND OTHERS AS THEY ARRIVE AT THE SMALL TABLES...AND WE'LL PUT THEIR COATS ON THE BED....
www.lloseng.com

Reality



Questions?

Homework:

- **Read**
  - Chapter 1 Introduction Software Engineering
  - Chapter 2 Review Object Orientation