Advanced Topics of Software Engineering (ASE)

Prof. Dr. Florian Matthes, Prof. Dr. Alexander Pretschner

Chapter 1. The context of software engineering

Department of Informatics
Technische Universität München, Germany

Module: IN2309, IN2126

Learning objectives

- Appreciate software engineering
  - ***Build complex software systems in the context of frequent change***

- Understand
  - how quality attributes affect the software architecture and conversely, how architectures influence these attributes in different domains
  - why there is a need to make quality trade-offs while considering different software architectures including component-based and service-oriented architectures
  - how to efficiently deliver the developed software system to the stakeholders

- Be able to apply
  - modeling techniques
  - system analysis and design by considering quality trade-offs
  - patterns, guidelines, and best-practices in software engineering
  - tools for system configuration, integration, and deployment

Organizational issues (1)

Module no	Studies	ECTS	Scope of the exam
IN2126	Masters in automotive software engineering	6	Chapter 1, 2, and 3
IN2309	Other than automotive software engineering	8	Everything that is covered in the lecture

- **Assumptions:**
  - You are a Master's student
  - You have taken Module IN0006 - Introduction to software engineering (EIST) or a similar course
  - You know Java, in particular object-oriented programming constructs
  - You have already experience in at least one analysis and design technique, preferred UML
- **Beneficial:**
  - You have had practical experience with a large software system
  - You have already participated in a larger software project
  - You have experienced major problems

Organizational issues (2)

- **Main lecture**:
  - Thursdays; 08:30 - 10:00; Interims Hörsaal 2
  - Fridays; 08:30 - 10:00; Interims Hörsaal 2

- **Lecture and exercise material** will be available on Moodle
  - Registration for the course in TUM Online is required
  - The lectures **will not be** recorded

- **Written examination**
  - Date - 17.02.2016; Time - TBD
  - Exam duration
    - IN2039 - 100 minutes (100 points)
    - IN2126 - 75 minutes (75 points)
  - Exam location – TBD

  - Must register for the exam in TUM Online
  - Language of examination – English only
  - Closed-book exam
    - English dictionary is allowed

Organizational issues (3)

- **Exercises**:

Group 1	Mondays	09:00 - 11:00	01.11.018
Group 2	Mondays	13:00 - 15:00	01.11.018
Group 3	Mondays	15:00 - 17:00	01.11.018
Group 4	Tuesdays	15:00 - 17:00	01.11.018
Group 5	Wednesdays	10:00 - 12:00	00.11.038
Group 6	Wednesdays	12:00 - 14:00	00.11.038

- **Exercise bonus**
  - To get the 0.3 bonus for the final grade students have to
    - pass the exam
    - show-up in all exercise classes (2 jokers)
    - hand-in final executable project
    - present the solution of any one exercise to the group at least once
    - All or nothing policy!
  - Not transferable to the next semesters
    - Applicable to **repetition** exam

What you will see

Structure of the lecture (1)

1. The context of software engineering

1.1. Introduction and overview
1.2. Factors affecting the design of a software system
1.3. Characteristics of software systems in different domains
1.4. Case studies from two domains (Two guest lectures!)

2. From requirements to system design

2.1. Software architecture
2.2. Software libraries and frameworks
2.3. Antipatterns in software engineering
2.4. Model-driven software engineering
2.5. Software product line engineering
2.6. Safety
2.7. Information security
2.8. Testability

Structure of the lecture (2)

3. Software architectures and their trade-offs

- 3.1. Introduction to distributed systems and middleware
- 3.2. Database-centric architectures
- 3.3. Message-oriented architectures
- 3.4. Object-oriented architectures
- 3.5. Component-based architectures
- 3.6. Service-oriented architectures

4. From source code to physical deployment

- 4.1. Introduction and historical perspective
- 4.2. Version control
- 4.3. Continuous integration
- 4.4. Continuous deployment
- 4.5. Virtual machines and containers
- 4.6. Software architectures for the cloud

Learning objectives of the exercise

Objectives

- Recap and apply theoretical aspects from lecture to practical problems
- Hands-on use of state-of-the-art technology
- Develop a medium-sized hybrid IS-ES distributed system
- Simulation of typical junior developer work assignments (e.g. code inspection, refactoring, maintenance, ...)

Caveat

You will need to get your hands dirty!

Assumptions

- Confident in standard Java (SE)
- Basic understanding of internet architecture
- Ability/endurance to work with complex tool chains and a medium-sized, non-trivial code base

Structure of the exercise (1)

- **Weekly exercise assignments**
  - Exercise sheets will be published on moodle every Friday
  - Exercise solutions will ***not*** be published

- **Procedure**
  - One student group presents solution of previous week
  - Tutor discusses theory question interactively with students
  - Tutor explains requirements for practical assignments
  - Students work on assignments under supervision of tutor
  - Assignments that could not be finished in-class are to be completed till next exercise sessions

- **First half of the semester:**
  - 50% Theory
  - 50% Small hands-on assignments

- **Second half of the semester:**
  - 25% Theory
  - 75% Project with pre-existing software system

Structure of the exercise (2)

Project

- Content
  - E-Ticket system mainly using Google technology
    - Android
    - Google App Engine
    - Raspberry Pie
    - JavaSE

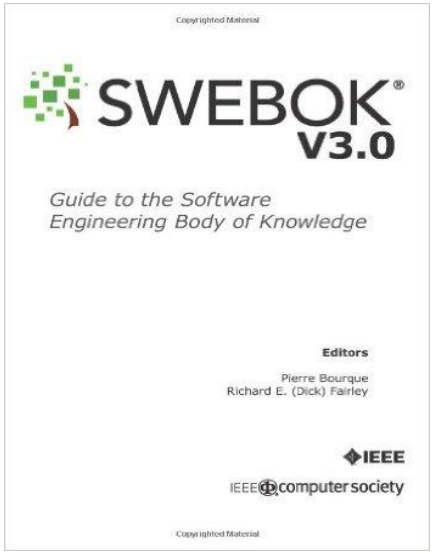
- Tasks
  - Understand and debug an existing codebase
  - Implement two pre-defined features
  - Implement one free-to-choose features

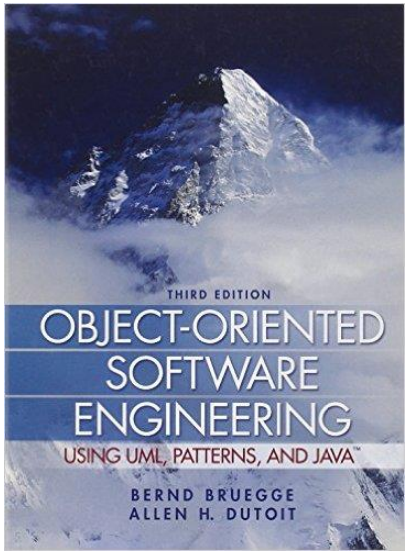
Out of scope for this lecture

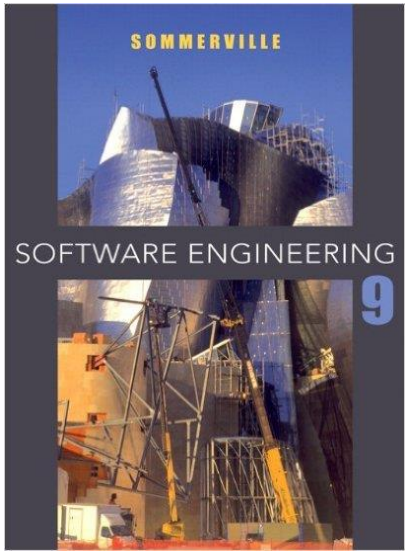
We do not cover the following topics in depth

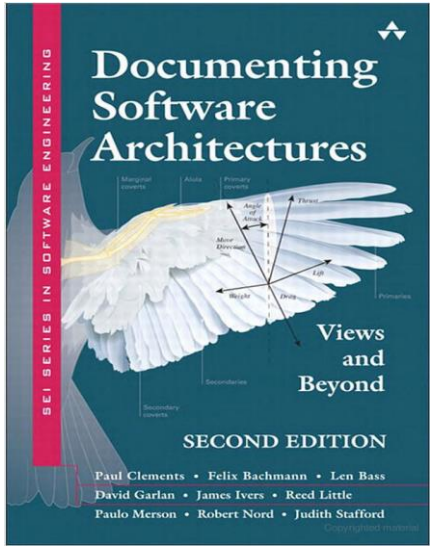
▪ Software engineering process	Introduction to software engineering (IN0006)
▪ Requirements engineering	Requirements engineering (IN2198)
▪ Project management	Project organization and management (IN2083)
▪ Design patterns	Patterns in software engineering (IN2081)
▪ Software testing	Advanced topics of software testing (IN2256)
▪ Software quality management	Software quality management (IN3050)
▪ Legal, social and economical aspects	Software Engineering in der Industriellen Praxis (IN2235), SEBA Bachelor (IN2085)

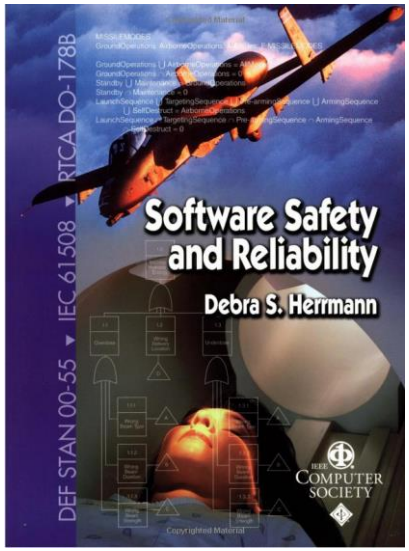
Reference books

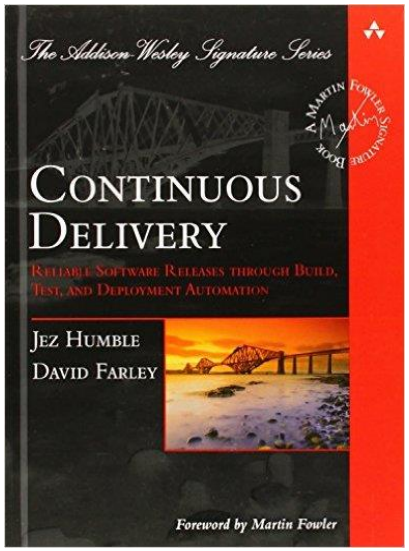












Additional references

- "Design rules: The power of modularity." Baldwin, C.Y. and Clark, K.B. (2000).
- "The Pragmatic Programmer: From Journeyman to Master." Hunt A. and Thomas D. (2000).
- "Patterns of enterprise application architecture" Fowler M. (2002).
- "Pattern-oriented software architecture volume 1: a system of patterns" Schmidt, D., Meunier R., Stal, M., Rohnert, H. and Buschmann, F. (1996).
- "Software Architecture in Practice." Bass L. and Clements P. (2012)
- "What is enterprise ontology?" Dietz J.L.G. (2006)
- "Object-oriented design with applications" Grady B. (1991).
- "Software Product Line Engineering." Pohl, K., Böckle G. and Linden F. (2005).
- "AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis." The Upstart Gang of Four (1998).
- " Software Product Quality Control." Wagner S. (2013)
- "Continuous integration" Fowler M. (2006).
- "The deployment production line" Jez H., Read C. and North D. (2006).
- "Softwaretechnik: Praxiswissen für Software-Ingenieure" Siedersleben, J. (2003).
- "Getting Started - About Version Control"

This is a recap.
These topics have been extensively covered in EIST

What is software engineering? (1)

"Study of the principles and methodologies for developing and maintaining software systems."

["Perspectives on software engineering." Zelkowitz. (1978)]

"Methods and techniques to develop and maintain quality software to solve problems."

["Software engineering: methods and management." Pfleeger. (1990)]

"Software engineering is an engineering discipline which is concerned with all aspects of software production."

["Software engineering." Sommerville. (2010)]

What is software engineering? (2)

Software engineering is a collection of techniques, methodologies and tools that help with the production of

- a *high quality* software system
- with a given *budget*
- before a given *deadline*

while *change* occurs.

- **A problem solving activity**
  - *Analysis*:
    - Understand the nature of the problem and break the problem into pieces
  - *Synthesis*:
    - Put the pieces together into a large structure
    - For problem solving we use *techniques*, *methodologies* and *tools*.

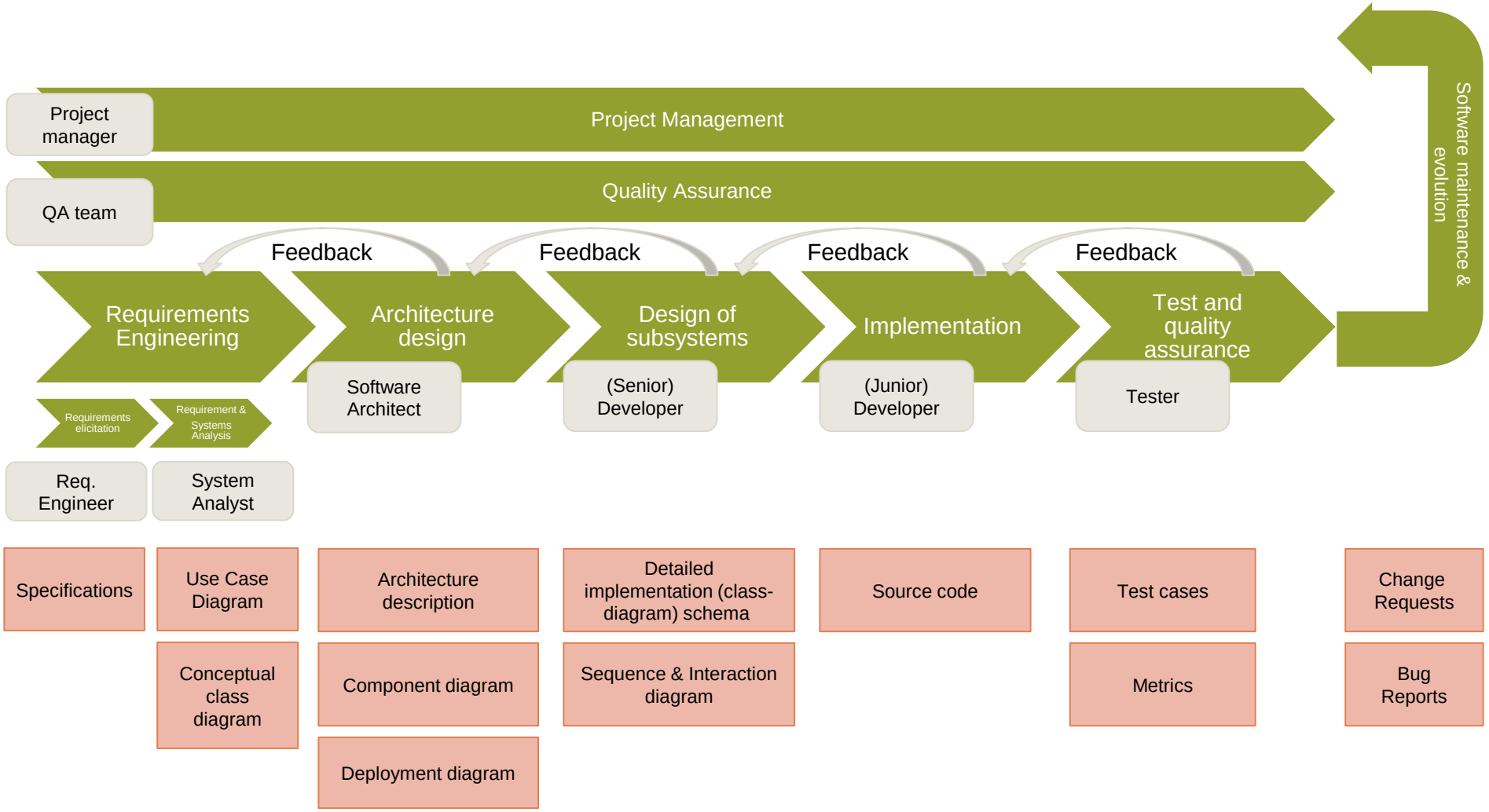
["Object-oriented software engineering using uml, patterns, and java." Bruegge B. and Dutoit A.H. (2009)]

Techniques, Methodologies and Tools

- **Techniques:**
  - Formal procedures for producing results using some well-defined notation
- **Methodologies:**
  - Collection of techniques applied across software development and unified by a philosophical approach
- **Tools:**
  - Instruments or automated systems to accomplish a technique
  - Integrated Development Environment (IDE)
  - Computer Aided Software Engineering (CASE)

["Object-oriented software engineering using uml, patterns, and java." Bruegge B. and Dutoit A.H. (2009)]

Software engineering activities, roles and artifacts

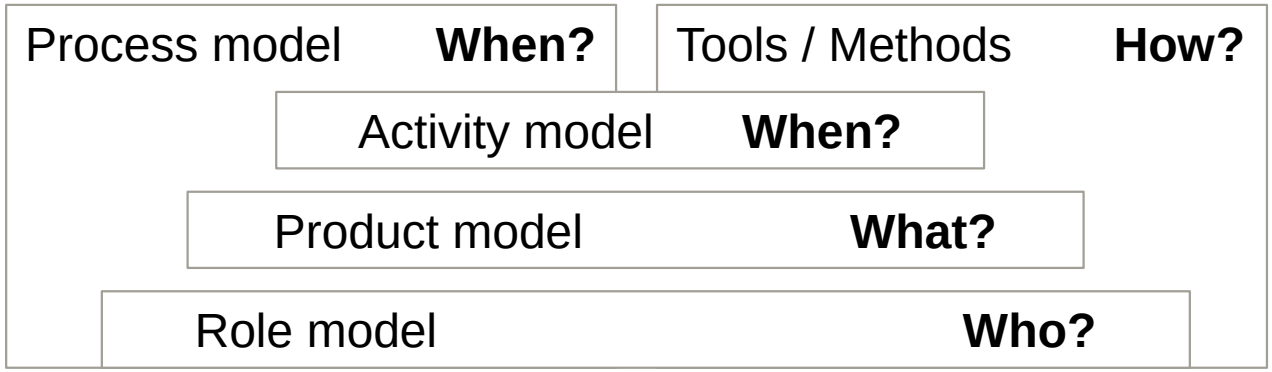


The process model

Definition

- A process model describes the systematic, engineering-based, and quantifiable approach to solve a particular class of repeatable problems.
- It is an abstract representation of the software process.

Structure



The process models

There exist many process models. For example:

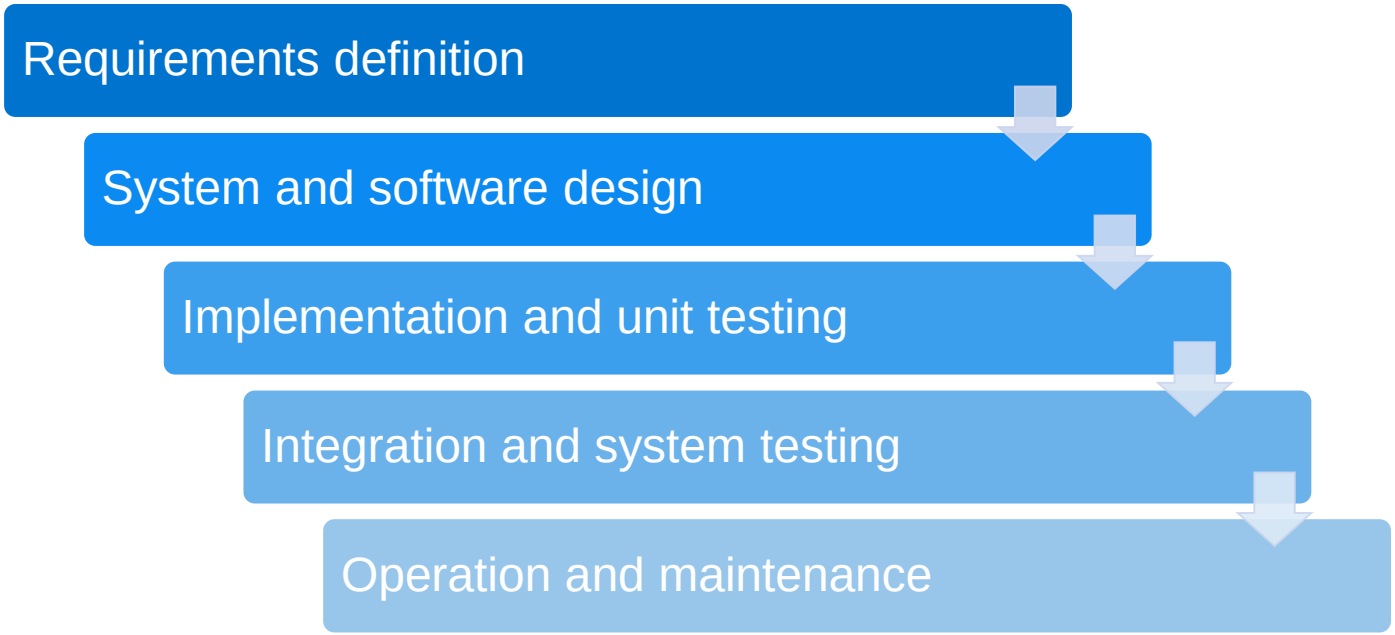
- Waterfall model (phase-oriented, sequential)
- Iterative model (phase-oriented, multiple pass)
- Incremental model (development in expansion stages)

Areas of conflict

- Heavyweight process (process-centric, rigid)
- Lightweight process (code-centric, agile)

The waterfall model

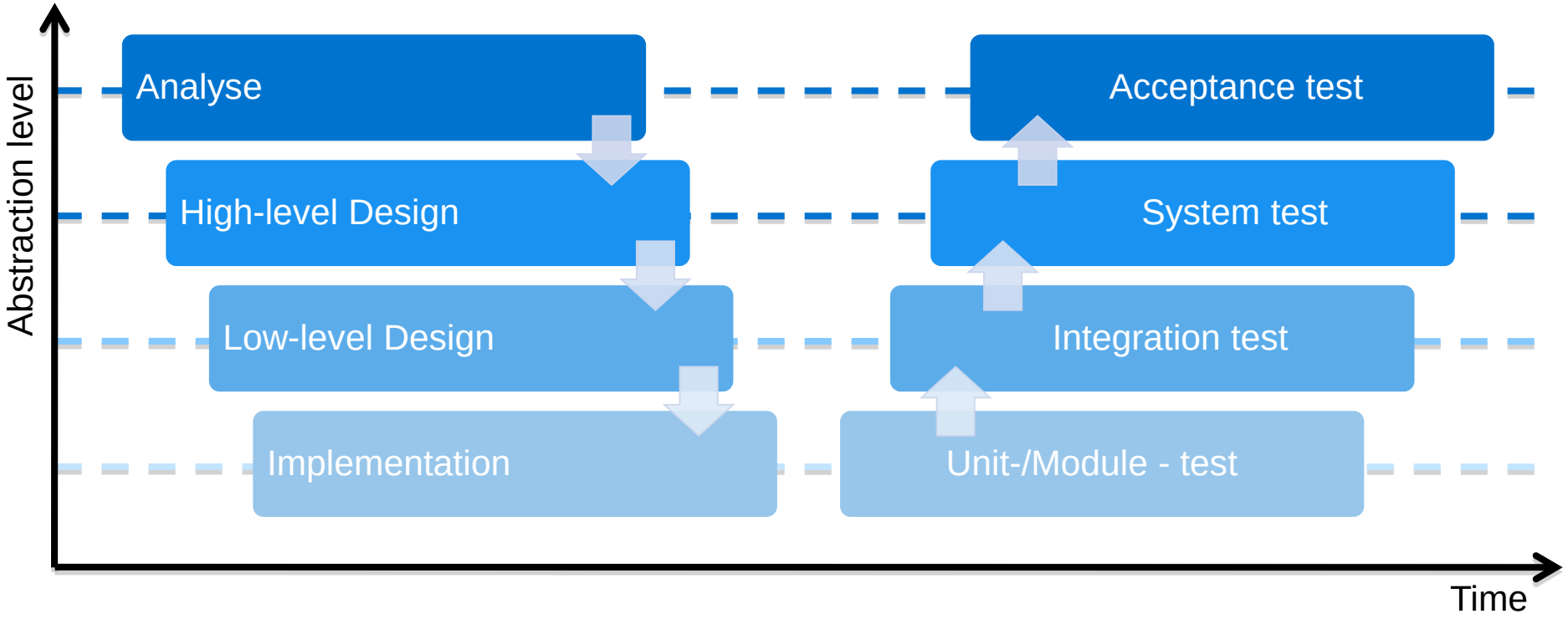
- Classical / conventional process model (goes back to Royce, 1970)
- Inflexible (sequential procedure is enforced)
- Risky (errors are found late and are expensive)



["Software Engineering." Ian Somerville. (2010)]

The V-model

- Originally developed in Germany for government defense projects
- Loosely based on the waterfall model
- Adapted and developed by MoD and BMI to the V-Model 92/97 / XT (from the early 1980's)



["Software Engineering." Ian Somerville. (2010)]

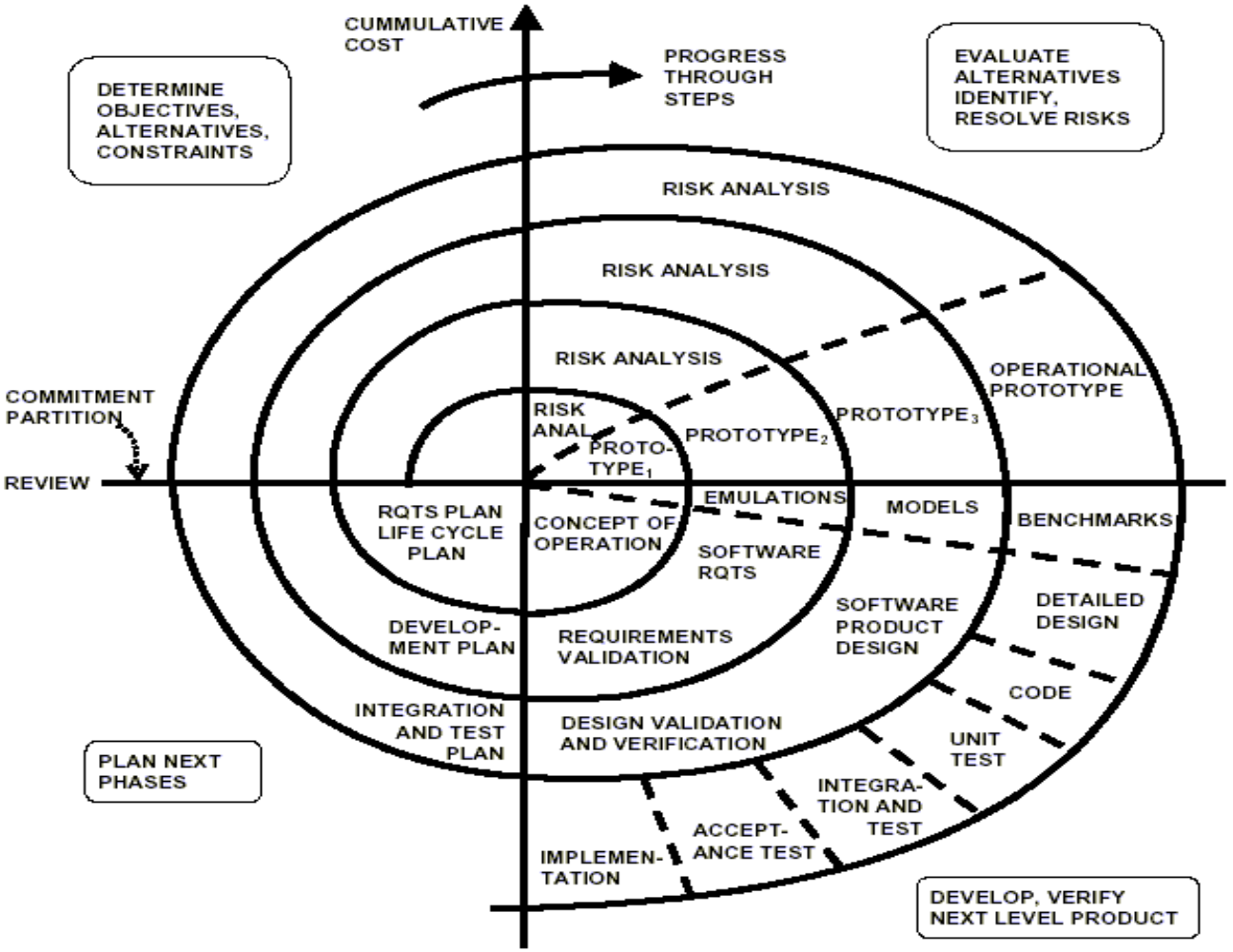
The spiral model

These include:

- Incremental
- Iterative
- Evolutionary models

- Self-learning process

- Aim: *risk minimization*

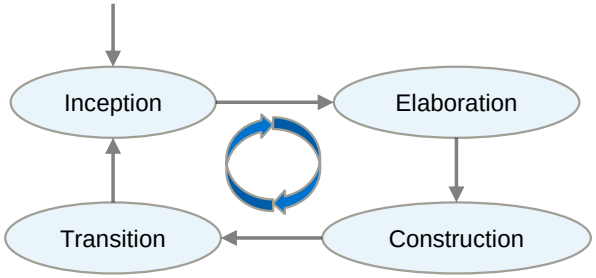


["A spiral model of software development and enhancement." Boehm B.W. (1988)]

The unified process

Unified software process / unified process: Iterative software lifecycle model

- Also referred to as unified process
- Emphasis on early construction of a software architecture
- Emphasis on early demonstrations of the system



Definitions

- **Phase:** Status of the software system.
  - 4 phases: Inception, Elaboration, Construction, Transition
- **Workflow:** Mostly sequential activity that produces artifacts
  - 7 workflows: Management, environment, requirements, design, implementation, assessment, deployment.
  - *5 artifact sets: Management set, requirements set, design set, implementation set, deployment set*
- **Iteration:** Repetition within a workflow.

Each unified process iteration is a software project.

["The unified software development process." Jacobsen I., Booch G. and Rumbaugh J. (1999)]

Processes in the unified process

Micro process: Policies and practices for building an ***artifact***

- Focus: Intermediate baselines with adequate quality and functionality as economically and rapidly as practical
- Same as "Process" in the IEEE 1074 Standard

Macro process: A set of micro processes and the dependencies among them

- Focus: Production of a software system within cost, schedule and quality constraints
- Also called: Life cycle model

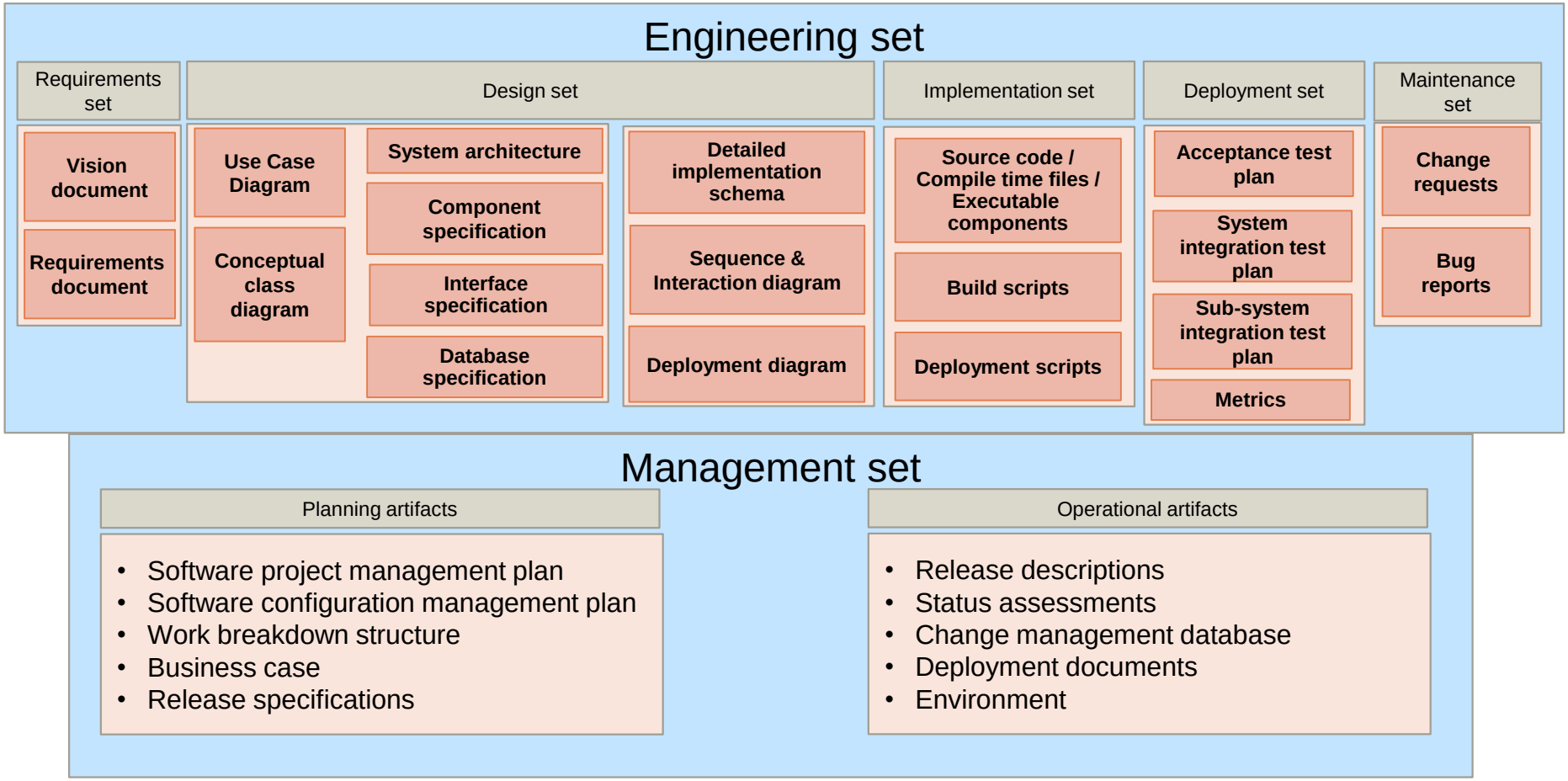
Meta process

- Focus: Organizational improvement, long-term strategies, and return on investment (ROI)
- Also called: Business process.

["The unified software development process." Jacobsen I., Booch G. and Rumbaugh J. (1999)]

Artifact sets in the unified process

Software artifact: a software work product produced by a process step; for e.g., an architecture design document is an artifact produced by the "design" step.



Representation of artifact sets

- **Management set**
  - *Goal*: Capture plans, processes, objectives, acceptance criteria
  - *Notation*: Ad hoc text, graphics, textual use cases.
- **Requirements set**
  - *Goal*: Capture problem in language of problem domain
  - *Notation*: Structured text, UML models
- **Design set**
  - *Goal*: Capture the engineering blueprints
  - *Notation*: Structured text, UML models.
- **Implementation set**
  - *Goal*: Capture the building blocks of the solution domain in human-readable format
  - *Notation*: Programming language
- **Deployment set**
  - *Goal*: Capture the solution in machine-readable format
  - *Notation*: Machine language.

Agile development

The aim is a leaner, less bureaucratic development process with

- fewer rules and
- a focus on working software that solves the customer's problem.

Values of agile software development

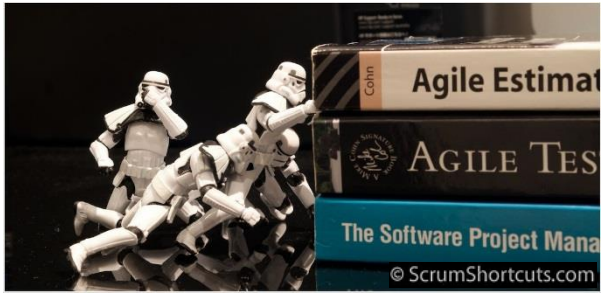
- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

The Agile Manifesto was signed in 2001.

[from the Agile Manifesto - www.agilemanifesto.org]

Agile development: Example: Scrum (1)

Scrum assumes that the software development process is too complex to be able to plan it in detail. Instead, scrum believes in self-organizing teams.



- Daily Scrum meeting is a central element
- Short duration (15 minutes)
- Stand-up meetings
- Each team member answers the following questions
  - What are the tasks you have completed since the last meeting?
  - What are the tasks you'll perform until the next meeting?
  - Are there problems that hinder your to-do tasks?

Scrum in German translates to "das Gedränge" and is also referred to the default situation in rugby.

Agile development: Example: Scrum (2)

The product requirements from the customers are collected in the **product backlog**.

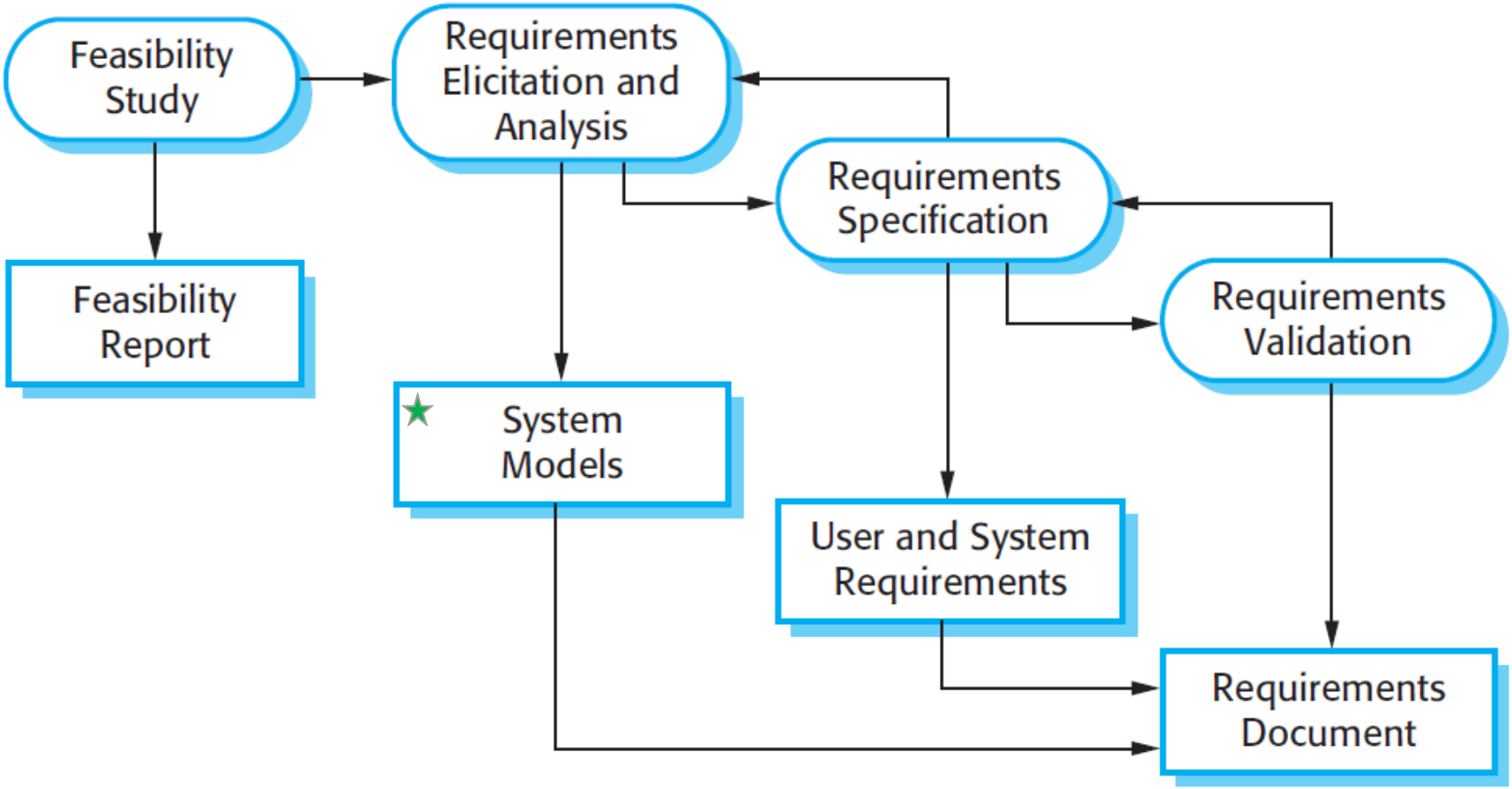
The development process is divided into **sprints**.

- A sprint is typically 1-4 weeks long.
- The tasks for the sprint are selected from the product backlog.

Scrum roles

- The **product owner** specifies firmly the common goal and prioritizes the tasks in the product backlog
- The **team** estimates the efforts and benefits of the tasks in the backlog and chooses what it wants to achieve by the end of the next sprint
- The **scrum master** is responsible for the scrum process. He is neither a member of the team nor the product owner. He oversees the division of roles and the rights of other stakeholders. His goal is to achieve a smooth process.

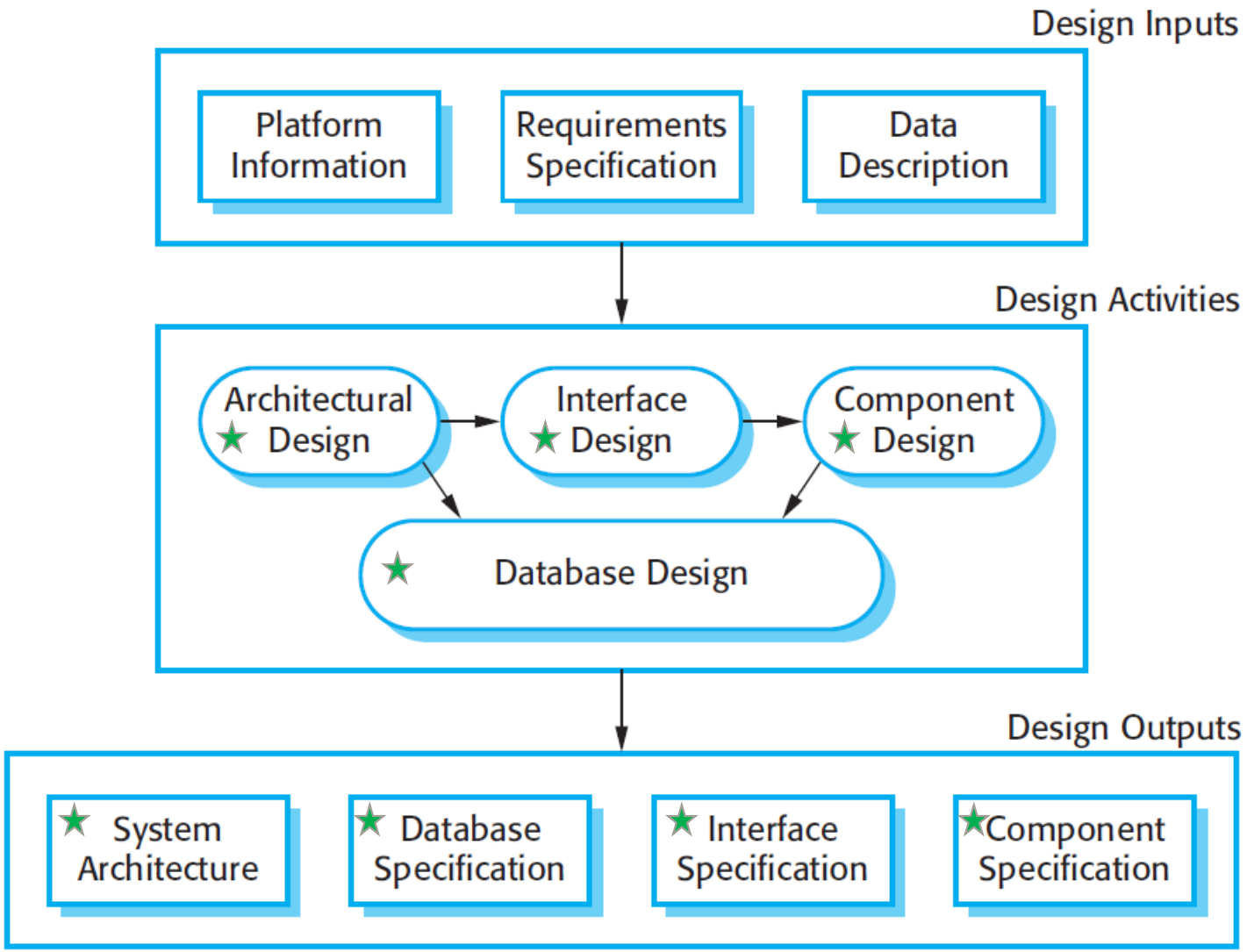
Artifacts generated in requirements engineering phase



- Please ignore the arrows between the activities!
- Focus on artifacts – rectangles

["Software Engineering." Ian Somerville. (2010)]

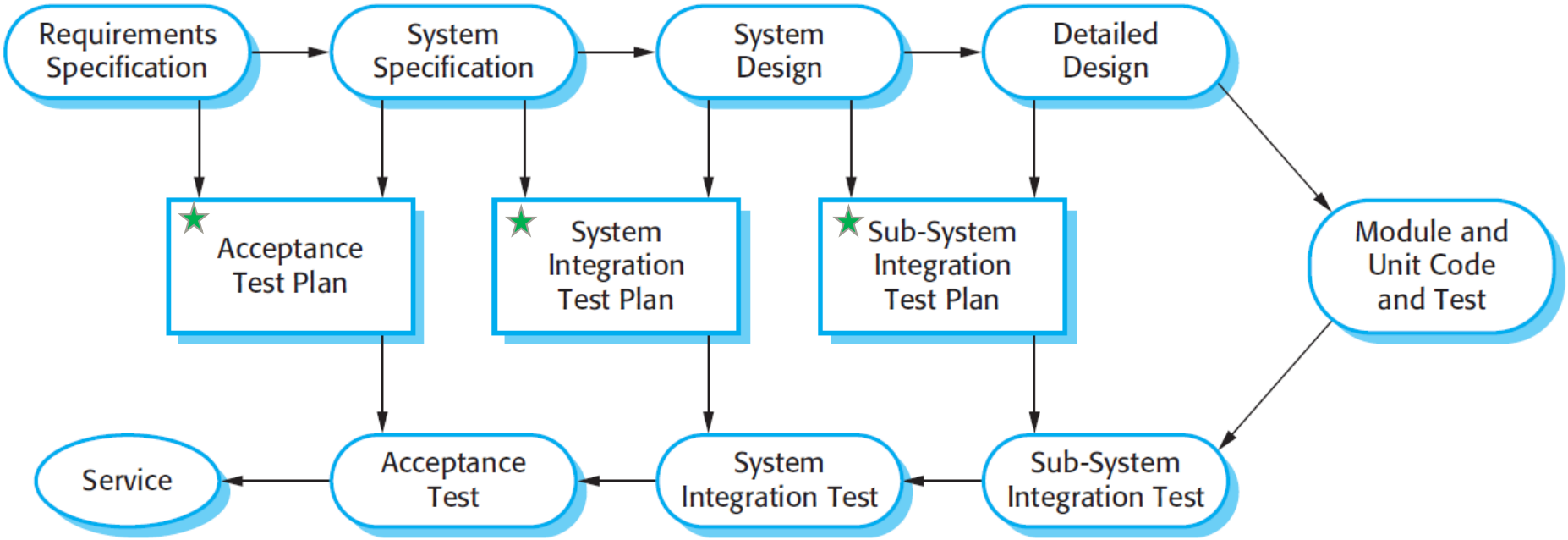
Artifacts generated in software design phase



Please ignore the arrows between the activities!

["Software Engineering." Ian Somerville. (2010)]

Artifacts generated in testing phases



- Please ignore the arrows between the activities!
- Focus on artifacts – rectangles

["Software Engineering." Ian Somerville. (2010)]

What is software quality?

"Software quality refers to the entire **characteristics of a software product** that influence its ability to **fulfill** specified **requirements** and stakeholder expectations."

[loosely based on Balzert]

"(Software) Quality is the **degree** to which a **set of inherent characteristics fulfills requirements**, meaning needs or expectations that are stated, generally implied or obligatory."

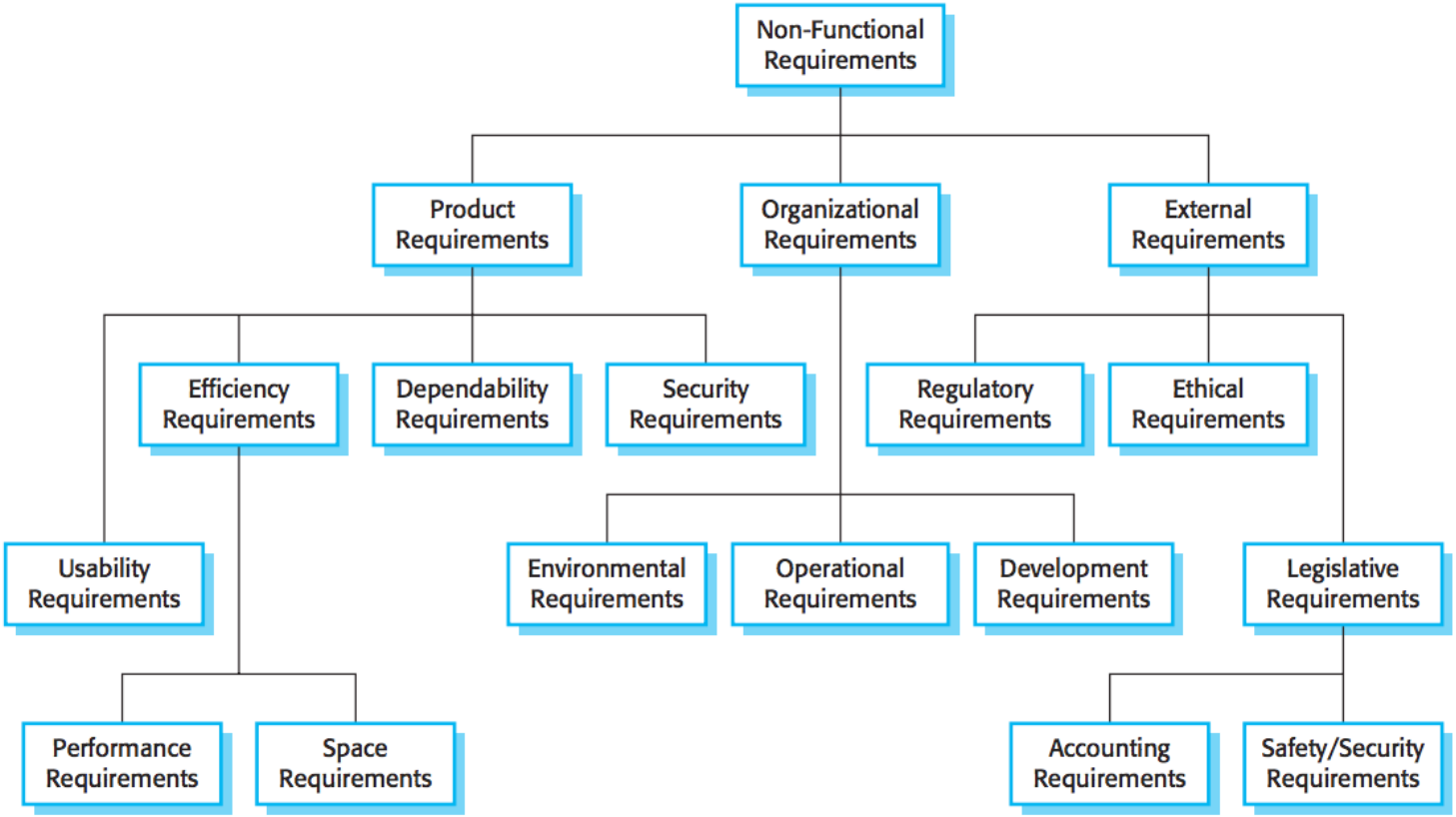
["ISO 9000 - Quality management" http://www.iso.org/iso/iso_9000]

Fuzzy term that refers to both: process and product. In contrast to cost and time often hard to measure!

Why is quality so important?

- Quality is the key factor for the product's success and customers' satisfaction.
- A product with all demanded functionality but bad quality (e.g. bad performance, bad security, …) is not likely to be accepted by customers.
- In many cases the quality demands are fixed (e.g. laws, contracts.)
- Solution attempts: various quality models, standards, certificates, and management / improvement approaches
  - Examples: ISO 9000 family, CMM, SPICE, ITIL, …
  - Rationale: mature processes lead to good products … always true?
  - Quality also needs to be addressed at the product level!

Quality requirements



["Software Engineering." Ian Somerville. (2010)]

Quality vs. non-functional requirements (NFRs)

- Quality and NFRs tightly related and often used as synonyms
  - but in general Quality ≠ NFR

- Both tackle the question of "how good is my software?", but
  - NFRs focus on qualifying aspects w.r.t. the product (e.g. timing behavior, availability, maintainability, …)
  - Quality has a wider scope since it also considers the process (e.g. time-to-market, …)
  - According to ISO 9126 *functionality* is a quality aspect but typically not a non-functional requirement

- Our understanding:
  Quality = Process Quality + (Product Quality = NFRs)

- Bottom line: remember that quality is more than just NFRs

Quality and trade-offs

- Typically there exists interdependencies between quality aspects
  - Positive: one quality aspect supports the other
    (e.g. maintainability vs. portability)
  - Negative: one quality aspect interferes with the other
    (e.g. security vs. efficiency)

- Not all quality aspects can be maximized simultaneously!

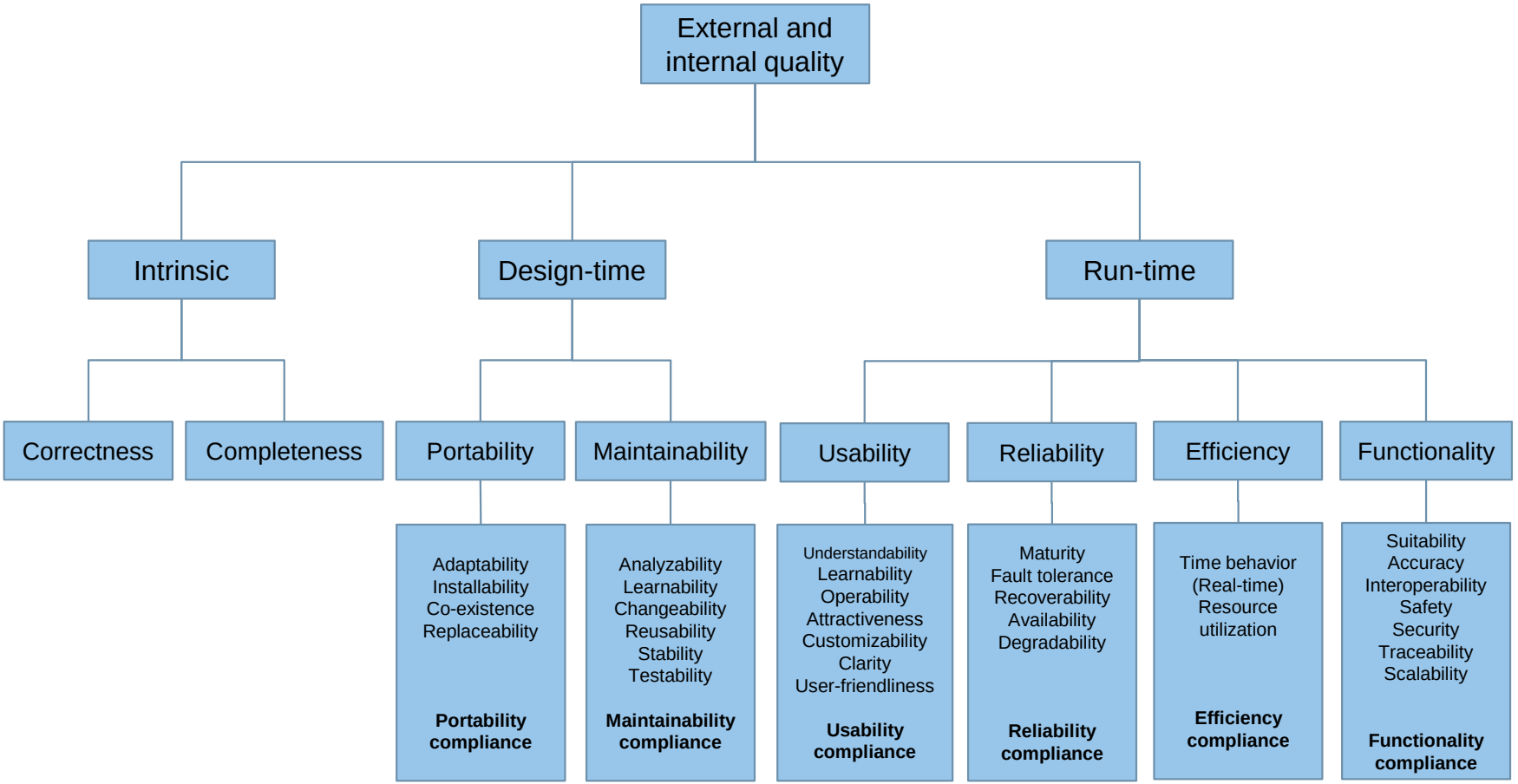
- Often *trade-offs* are necessary
  - Assess relevant quality aspects
  - Carefully assess interdependencies
  - Prioritize and balance aspects
  - Trade-offs and prioritization are highly situation and project dependent

Types of quality (1)

Quality categories relevant for and affected by the architecture

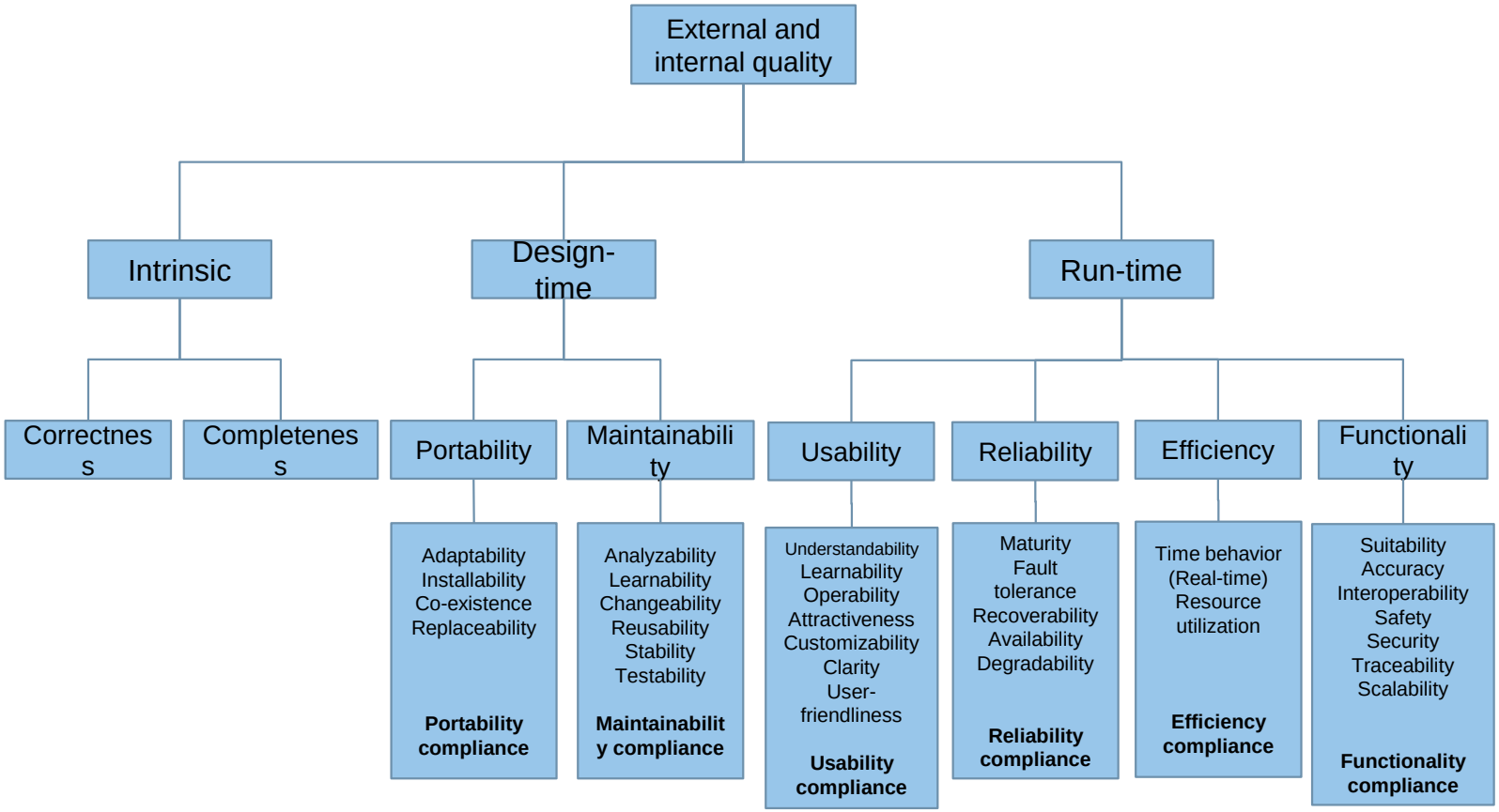
- **Intrinsic quality attributes**
  - Correctness (consistency) and completeness
- **Design time quality attributes**
  - Portability, maintainability, reusability, changeability, testability
- **Run time quality attributes**
  - Usability, reliability, efficiency, functionality, safety, security, performance, real-time, robustness, scalability, availability

Types of quality



[ISO/IEC 9126]

Types of quality



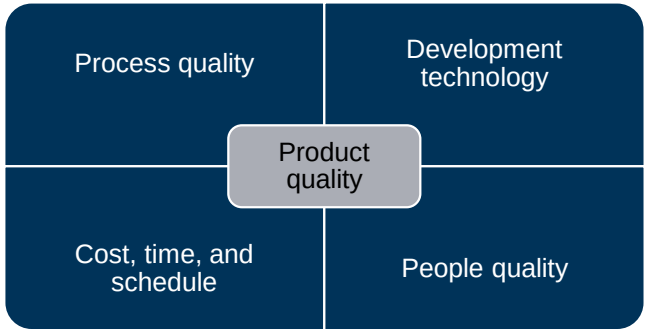
- These attributes refer both to the external as well as internal quality attributes.
- The quality attributes are applicable at multiple levels of granularity. For instance, one can define these attributes from a black-box or white-box perspective.

[ISO/IEC 9126]

Process and product quality attributes (1)

- The quality of the developed product is influenced by the quality of the production process

- The relationship between software process and software product quality is complex

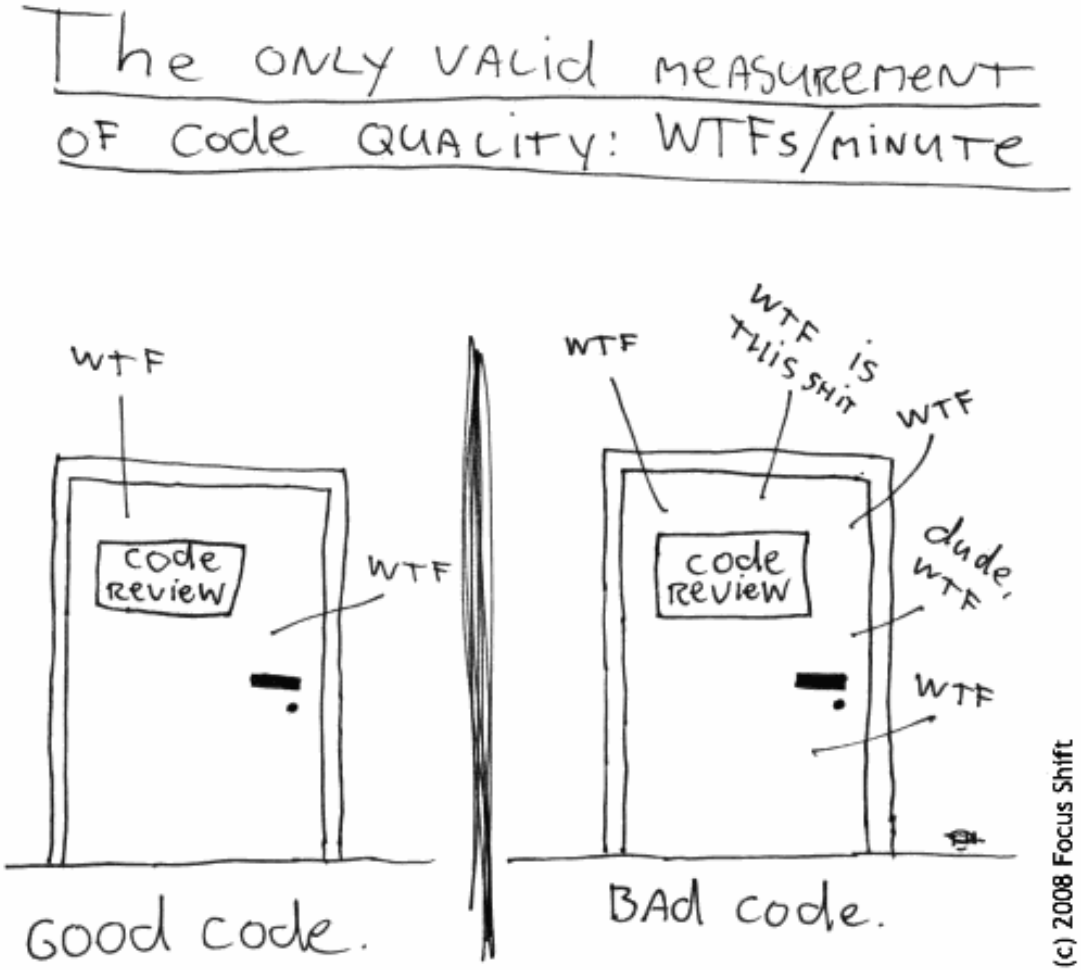


- Software is not manufactured but designed
- Software development is a creative rather than a mechanical process

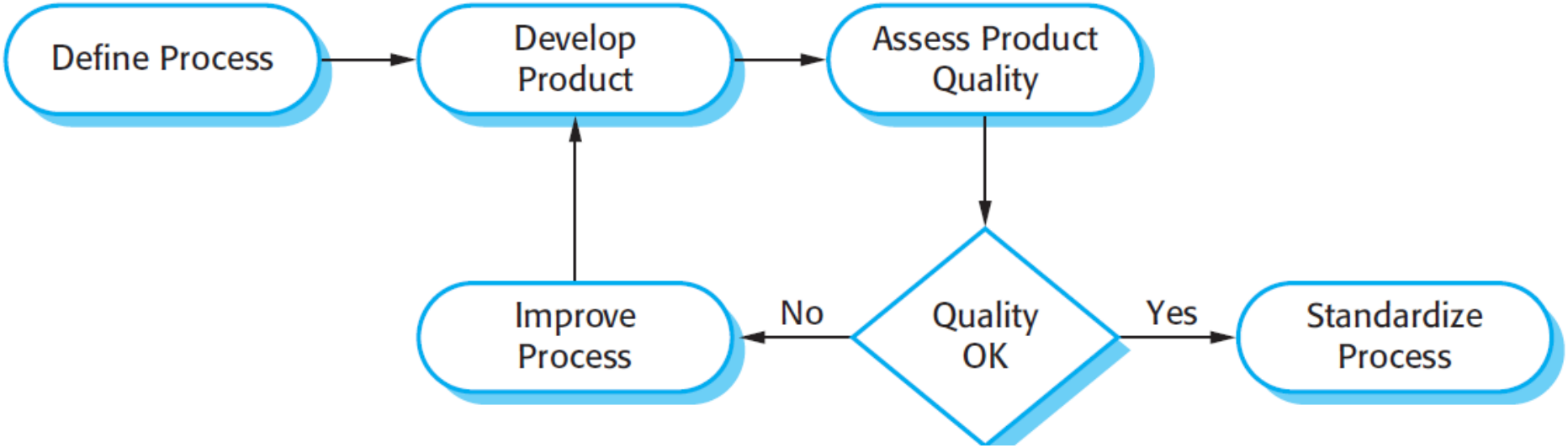
["Software Engineering." Ian Somerville. (2010)]

Process and product quality attributes (2)

- It is difficult to measure software quality attributes;
  - Maintainability cannot be measured without using the software for a long time



Process-based quality



["Software Engineering." Ian Somerville. (2010)]

Quality assurance and standards

- Standards are the key to effective quality management
- Standards capture wisdom that is of value to the organization (knowledge about the best/appropriate practice for the company.)
- Standards enable organizations to reuse their past experiences and to avoid previous mistakes.
- Standards provide a framework for defining what 'quality' means in a particular setting.

- Two types of standards for quality assurance process:
  - **Product standards** define characteristics that all components should exhibit; e.g. requirement document standard and coding standards
  - **Process standards** define how the software process should be enacted

["Software Engineering." Ian Somerville. (2010)]

Product and process standards

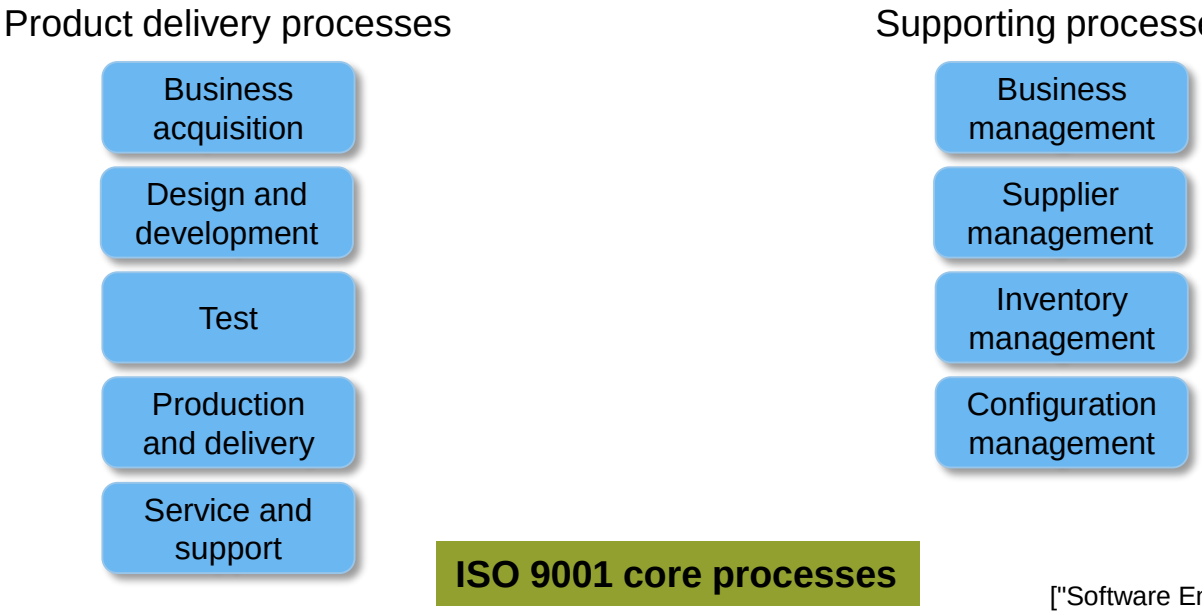
Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of code for system building
Procedure header format	Version release process
Programming style and conventions	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

["Software Engineering." Ian Somerville. (2010)]

ISO 9000 and ISO 9001

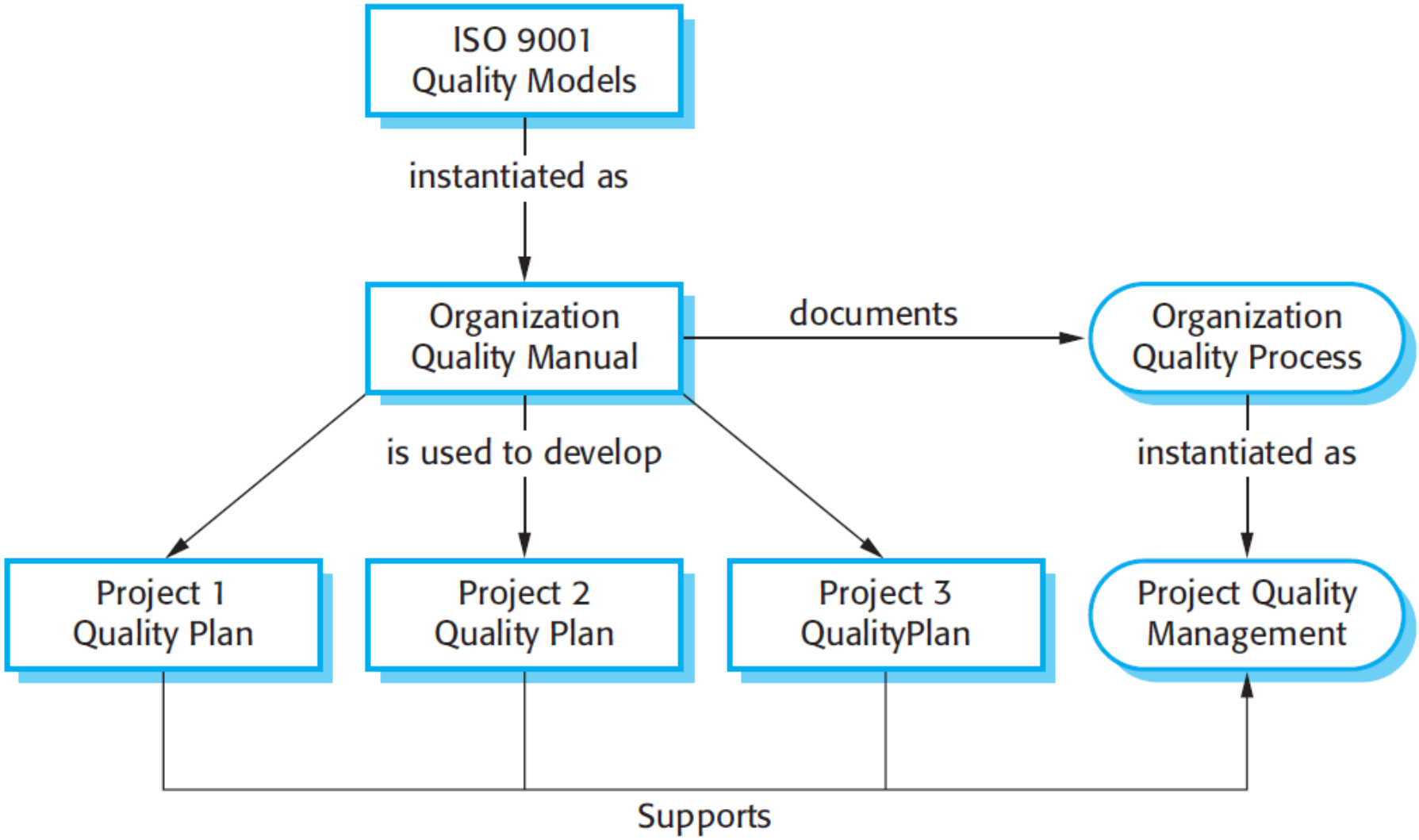
- An international set of standards for quality management
- Applicable to a range of organizations from manufacturing to service industries

- ISO 9001 (*generic*) applies to organizations that design, develop, and maintain products, including software
  - A framework for developing software standards
  - If an organization is to be ISO 9001 conformant, it must document how its processes relate to the nine core processes.



["Software Engineering." Ian Somerville. (2010)]

ISO 9001 and quality management



["Software Engineering." Ian Somerville. (2010)]

Quality management

Quality management

- Concerned with ensuring that the required level of quality is achieved in a software product.
- Involves defining appropriate quality standards and procedures and ensuring that these are followed.
- Should aim to develop a ‘quality culture’ where quality is seen as everyone’s responsibility.

Scope of quality management

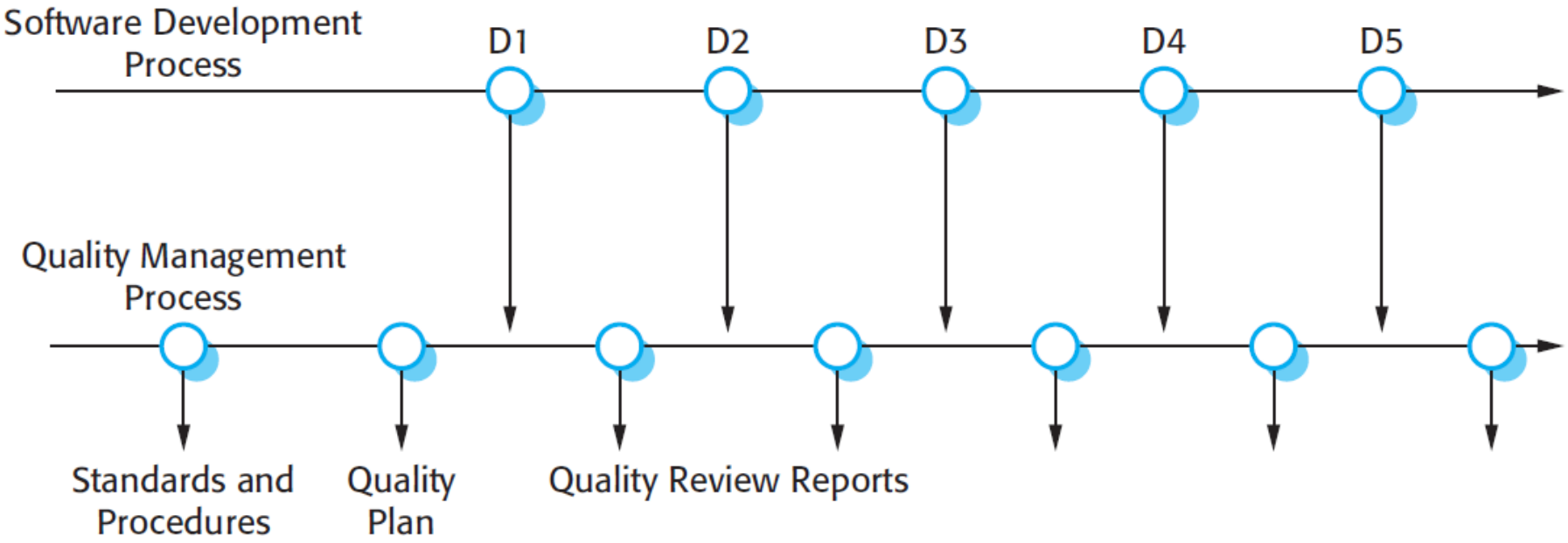
- Quality management is particularly important for large, complex systems.
- The quality documentation is a record of progress and supports continuity of development as the development team changes.
- For smaller systems, quality management needs less documentation and should focus on establishing a quality culture.

Quality management activities

- **Quality assurance**
  - Establish organisational procedures and standards for quality.
- **Quality planning**
  - Select applicable procedures and standards for a particular project and modify these as required.
- **Quality control**
  - Ensure that procedures and standards are followed by the software development team.
- Quality management should be separate from project management to ensure independence.

["Software Engineering." Ian Somerville. (2010)]

Quality management and software development



["Software Engineering." Ian Somerville. (2010)]

Quality standards – loopholes

- The ISO 9001 certification does not imply that the quality of the software produced by certified companies will be better than that of uncertified companies
  - For e.g. a company could define test coverage standards specifying that all methods in objects must be called at least once. This standard can be met by incomplete software testing, which does not run tests with different method parameters

- The ISO 9001 standard focuses on ensuring that the organization has quality management procedures in place and it follows these procedures

- The ISO 9001 certification defines quality to be the conformance to standards, and takes no account of the quality as experienced by users of the software

Companies that use agile development methods are rarely concerned with ISO 9001 certification.

["Software Engineering." Ian Somerville. (2010)]

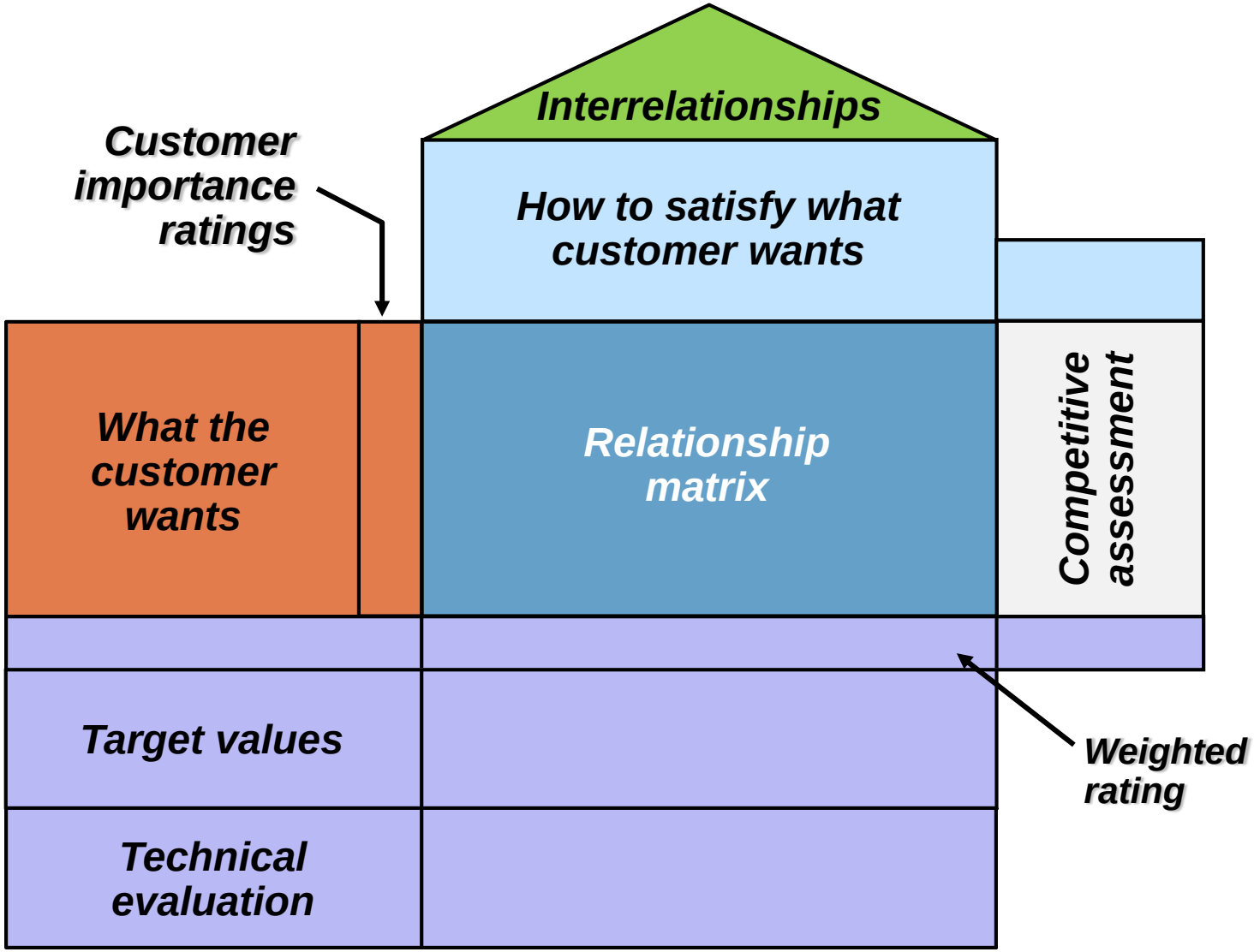
The house of quality - background

- A graphical tool that links customer needs to product capabilities
- It is a conceptual map that provides the means for inter-functional planning and communications
- Enables to understand what customers mean by quality and how to achieve it from an engineering perspective

- Should be employed at the beginning of every project
- Customer requirements should be translated into measurable design targets
- It can be applied to the entire problem or any sub-problem
- First worry about ***what*** needs to be designed then ***how***

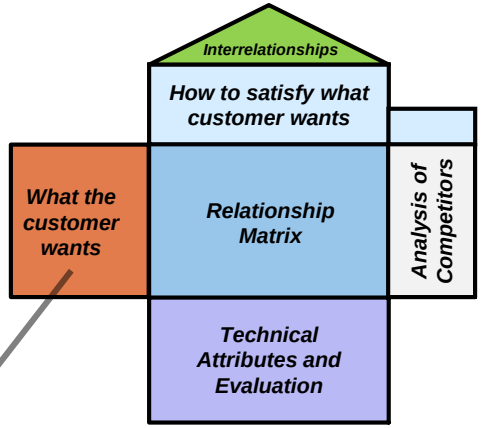
- It takes time to complete

The house of quality



Building the house

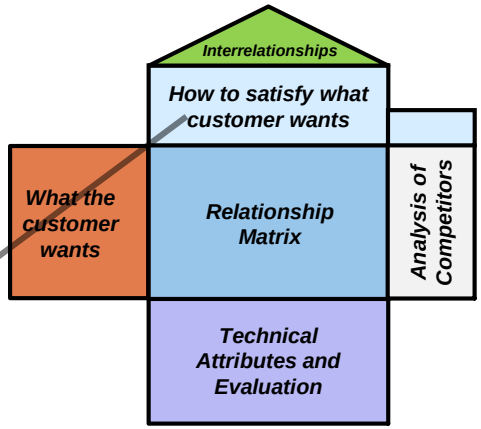
1. What do customer want?



What the customer wants	***Customer importance rating (5 = highest)***
Android OS	3
Long battery life	4
Reliable	5
Easy to use	2
Upload pictures to the cloud	1

Building the house

2. How to satisfy what customer wants?



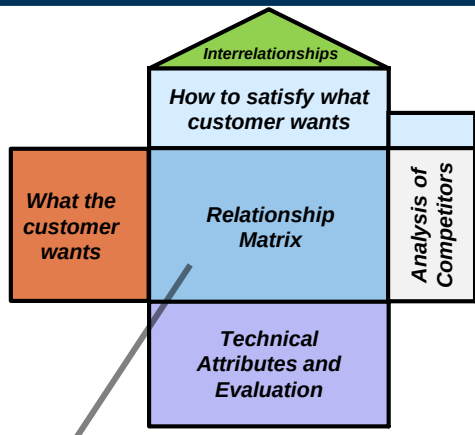
- Low electricity consumption
- OS support
- Wireless capability
- High resolution
- Auto focus
- Ergonomic design

Building the house

3. Relationship matrix

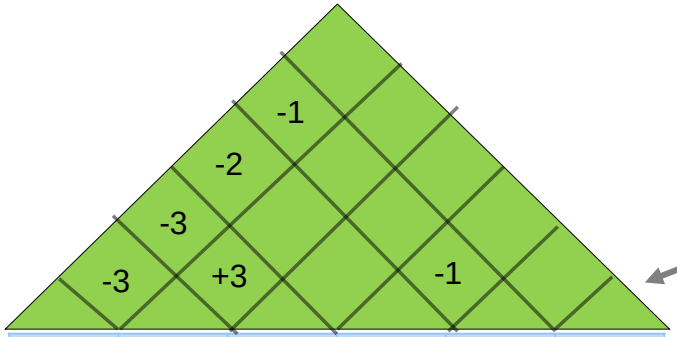
- High relationship (6)
- Medium relationship (3)
- Low relationship (1)

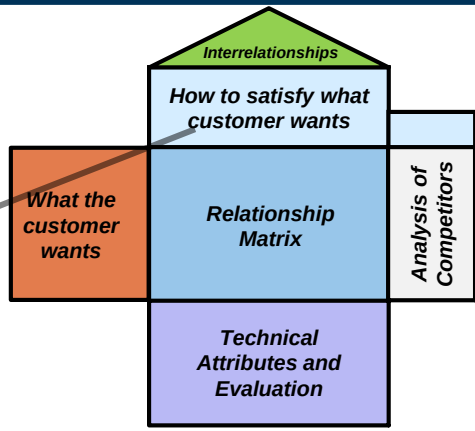
		Low electricity consumption	OS support	Wireless capability	High resolution	Auto focus	Ergonomic design
Android OS	3	•	◉	○			
Long battery life	4	◉	•	◉	○	•	
Reliable	5	○			○	○	
Easy to use	2					○	◉
Upload pictures to the cloud	1		○	◉			



Building the house

4. Relationships between the things we can achieve





Low electricity consumption

OS support

Wireless capability

High resolution

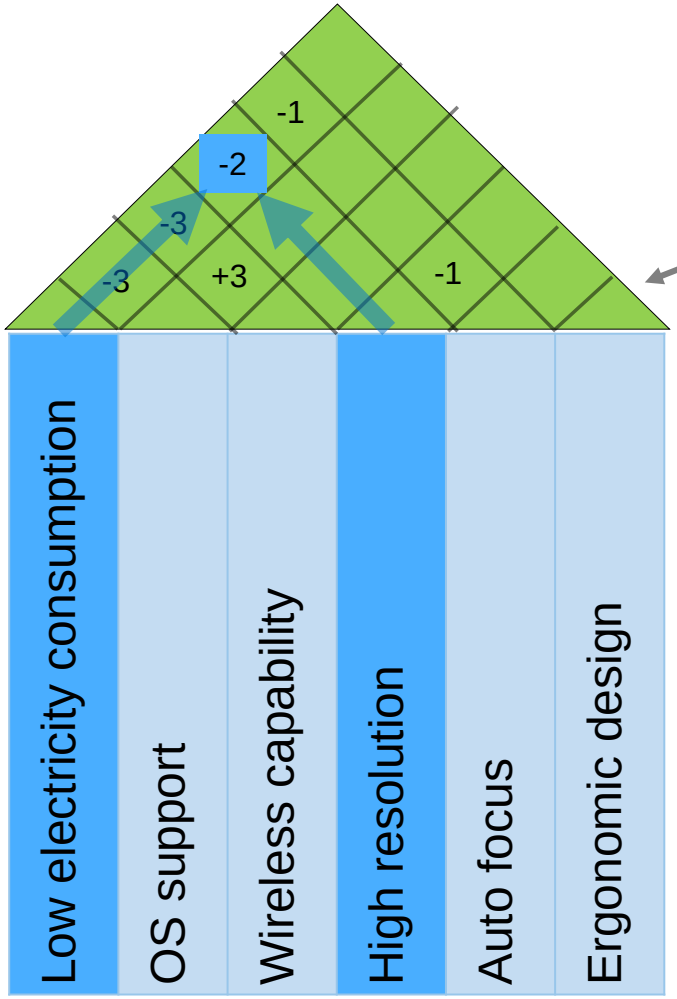
Auto focus

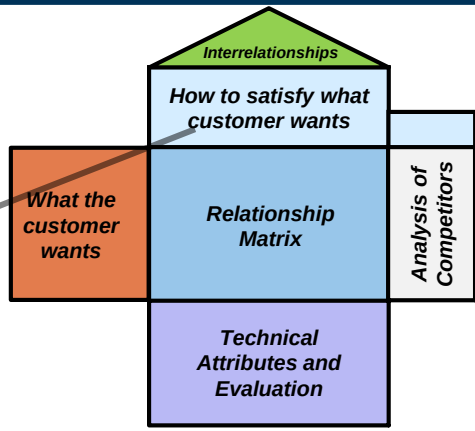
Ergonomic design

+ positive correlation
- negative correlation/trade-offs
+3 ---- highly positively correlated
0/empty block ---- no correlation
-3 ---- highly negatively correlated

Building the house

4. Relationships between the things we can achieve





- negative correlation example
  - High resolution is negatively correlated to the electricity consumption

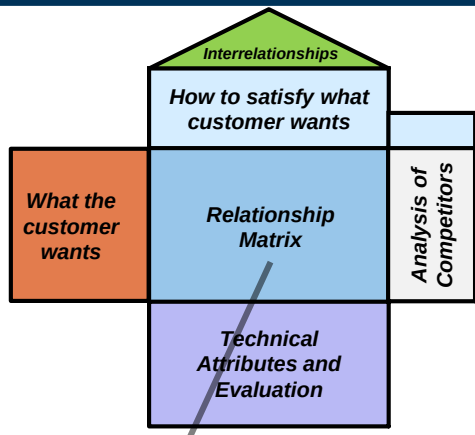
Building the house

5. Compute the weighted ratings

- High relationship (6)
- Medium relationship (3)
- Low relationship (1)

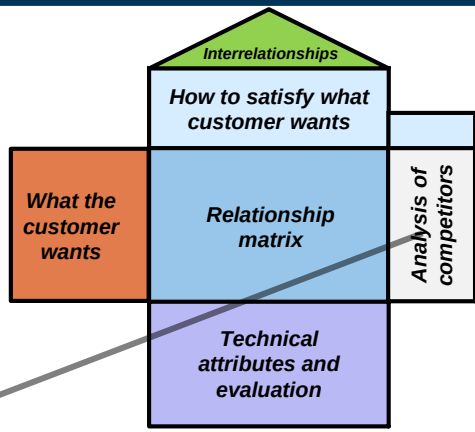
		Low electricity consumption	OS support	Wireless capability	High resolution	Auto focus	Ergonomic design
Android OS	3	•	◉	●			
Long battery life	4	◉	•	◉	●	•	
Reliable	5	●			●	●	
Easy to use	2					●	◉
Upload pictures to the cloud	1		●	◉			
Our importance ratings		42	25	39	27	25	12

3*1 + 4*6 + 5*3



Building the house

6. Analysis of competitors

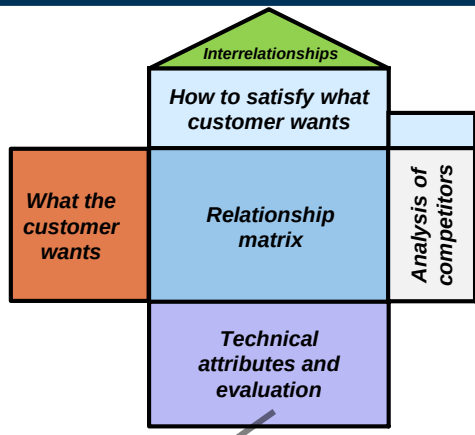


				Company A	Company B
Android OS	3			G	P
Long battery life	4	•		G	P
Reliable	5	○		F	G
Easy to use	2	○	◉	G	P
Upload pictures to the cloud	1			P	P
Our importance ratings		25	12		

G - good
F - fair
P - poor

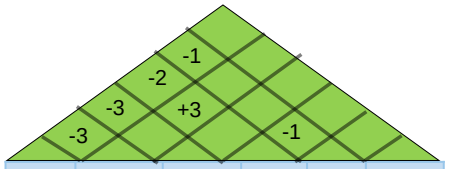
Building the house

7. Technical attributes and evaluation



Target values (Technical attributes)		0.5A	Android	802.11	38.3 megapixels	Failure 1 per 10,000	Panel ranking
Technical evaluation	Company A	0.7	no	yes	48	ok	G
	Company B	0.6	yes	no	28	ok	F
	Us	0.5	Yes	yes	32	ok	G

The completed house of quality



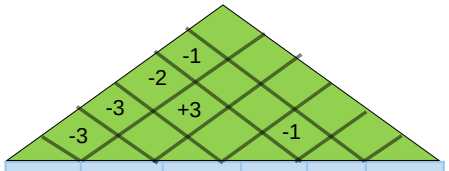
		Low electricity consumption	OS support	Wireless capability	High resolution	Auto focus	Ergonomic design	Company A	Company B
Android OS	3	●	◉	○				G	P
Long battery life	4	◉	●	◉	○	●		G	P
Reliable	5	○			○	○		F	G
Easy to use	2					○	◉	G	P
Upload pictures to the cloud	1		○	◉				P	P
Our importance ratings		42	25	39	27	25	12		
Target values (Technical attributes)		0.5A	Android	802.11	38.3 megapixels	Failure 1 per 10,000	Panel ranking		
Technical evaluation	Company A	0.7	no	yes	48	ok	G		
	Company B	0.6	yes	no	28	ok	F		
	Us	0.5	Yes	yes	32	ok	G		

Key to roof/correlation matrix symbols
+ positive correlation
- negative correlation/trade-offs
+3 ---- highly positively correlated
0/empty block ---- no correlation
-3 ---- highly negatively correlated

◉ High relationship
○ Medium relationship
● Low relationship

G - good
F - fair
P - poor

The completed house of quality



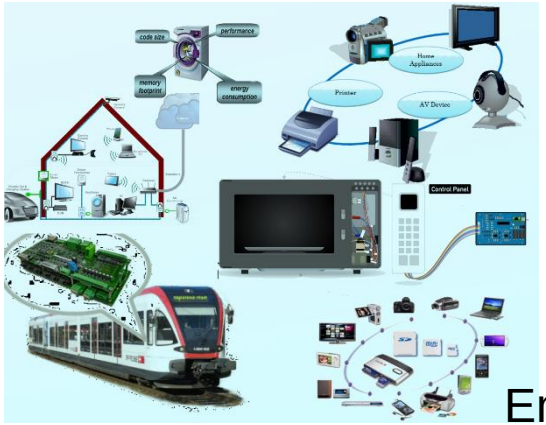
Key to roof/correlation matrix symbols
+ positive correlation
- negative correlation/trade-offs
+3 ---- highly positively correlated
0/empty block ---- no correlation
-3 ---- highly negatively correlated

		Low electricity consumption	OS support	Wireless capability	High resolution	Auto focus	Ergonomic design	Company A	Company B
Android OS	3	●	◉	○				G	P
Long battery life	4	◉	●	◉	○	●		G	P
Reliable	5	○			○	○		F	G
Easy to use	2					○	◉	G	P
Upload pictures to the cloud	1		○	◉				P	P
Our importance ratings		42	25	39	27	25	12		
Target values (Technical attributes)		0.5A	Android	802.11	38.3 megapixels	Failure 1 per 10,000	Panel ranking		
Technical evaluation	Company A	0.7	no	yes	48	ok	G		
	Company B	0.6	yes	no	28	ok	F		
	Us	0.5	Yes	yes	32	ok	G		

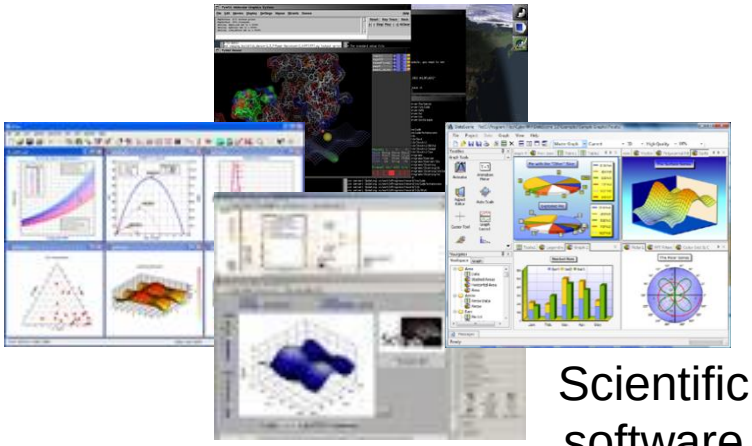
● High relationship
○ Medium relationship
● Low relationship

G - good
F - fair
P - poor

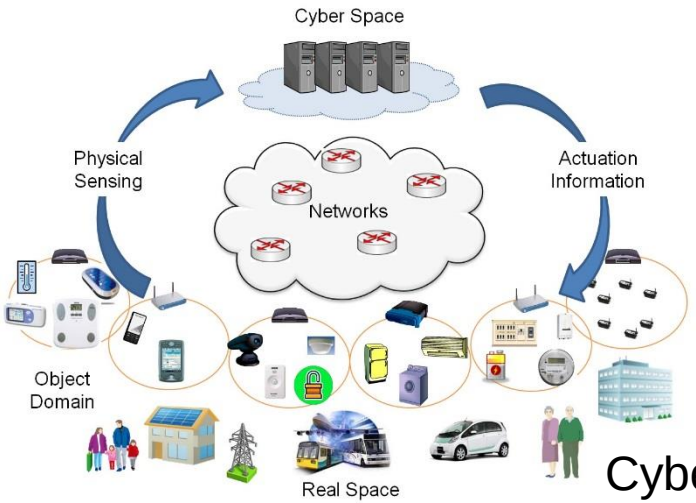
Multiple application domains



Embedded systems



Scientific software systems



Cyber-physical systems

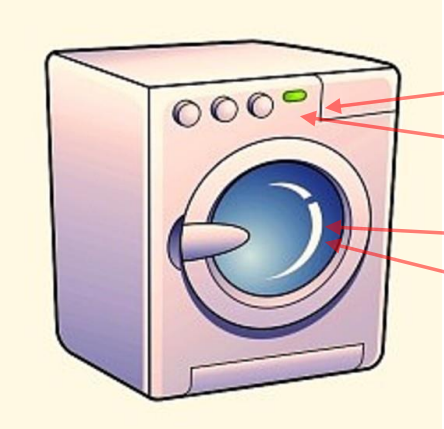
Information systems

Embedded systems (ES)

"An Embedded Computer System: A computer system that is part of a larger system and performs some of the requirements of that system; for example, a computer system used in an aircraft or rapid transit system."

[IEEE, 1992]

- Emphasis on software for devices that interact with their environment
- Communication with sensors and actuators
- Basically every electronic device nowadays is an embedded system

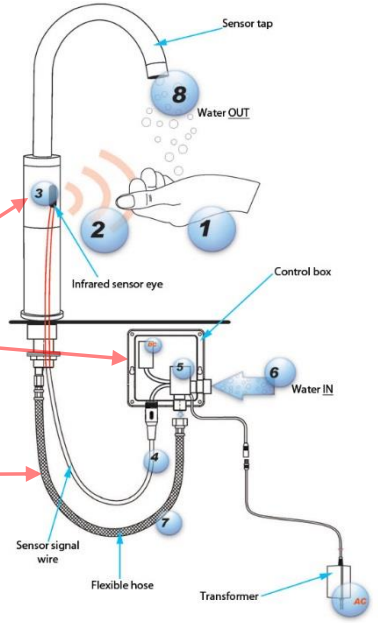


User interface

Processor

Sensors

Actuators



Characteristics of embedded systems

Processing: ability to process the analog/digital signals

- intricate control flows in control loops

Communication: ability to transfer information from/to the outside world

- communication interfaces is critical as it affects the final price of the product

Storage: ability to preserve the temporary information

- less data in terms of amount and complexity

User interface: in many embedded systems the user interface consists of a few buttons and/or LEDs; in others, it uses the user interface of a host system

The same functionality (e.g., the ability to acquire still images via a CCD sensor) can be optimized in radically different ways (e.g., a digital camera or a cell phone or a digital camcorder.)

["Embedded Systems Design for High-Speed Data Acquisition and Control." Emilio M.D.P. (2014)]

Characteristics of embedded systems

- Must be ***dependable***:
  - Reliability: R(t) = probability of system working correctly provided that it was working at t=0
  - Maintainability: M(d) = probability of system working correctly d time units after error occurred.
  - Availability: probability of system working at time t
  - Safety: no harm to be caused
  - Security: confidential and authentic communication

- Must be ***efficient***:
  - Energy efficient
  - Code-size efficient (especially for systems on a chip)
  - Run-time efficient
  - Weight efficient
  - Cost efficient
  - Higher emphasis on effects of concurrency

- Dedicated towards a certain application: Knowledge about behavior at design time can be used to minimize resources and to maximize robustness

- Higher emphasis on effects of concurrency

- Higher emphasis on optimization (sparse resources)

Characteristics of embedded systems

Quality attributes	Embedded system
Maintainability	+/-
Reliability	+
Availability	+
Safety	+
Security	+
Code-size efficiency	+
Energy efficiency	+
Run-time efficiency	+
Physical constraints (weight)	+
Cost	+
Predictability	+
Deadline dependent	+
Quality of service	+
Functionality	+
Usability	-/+
Performance	+
Supportability	-/+
Scalability	-
Recoverability	-
Transaction support	-

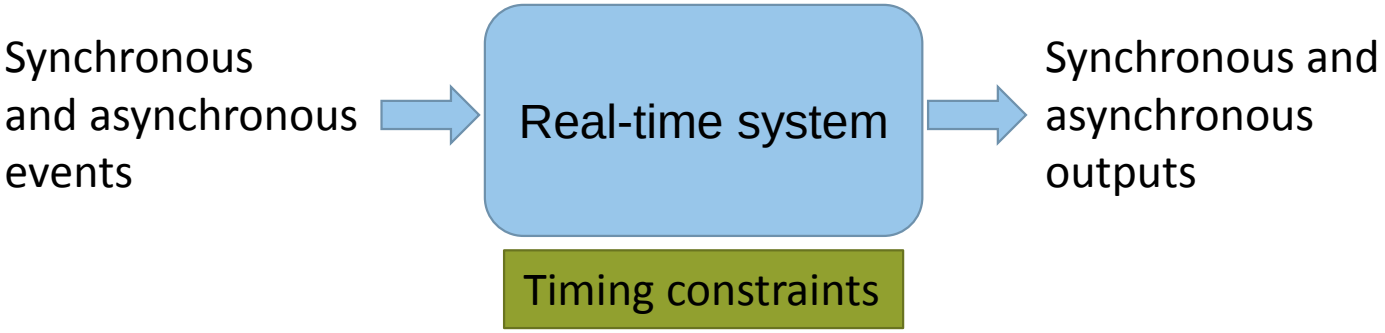
Dependability

Efficiency

Real-time

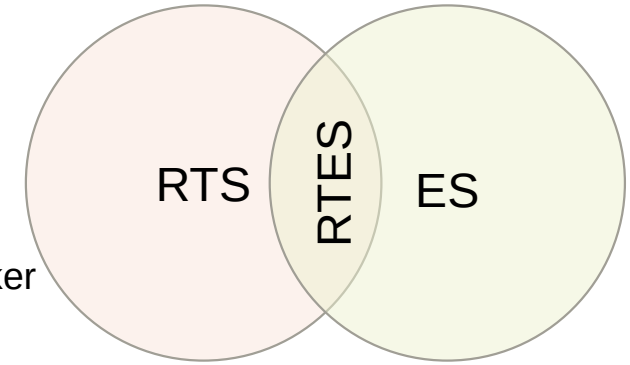
["Embedded Systems Design for High-Speed Data Acquisition and Control." Emilio M.D.P. (2014)]

Real-time system (RTS)



- The overall correctness of the system depends on both, the functional correctness and the timing correctness
- It has a substantial knowledge of the system it controls and the applications running on it
- **Deadline dependent**
- **Predictability** is very important

- RTS – Railway monitoring and scheduling
- ES – Mobile devices
- Real-time embedded systems (RTES) – Heart pacemaker



Reactive systems

"A reactive system is one which is in continual interaction with is environment and executes at a pace determined by that environment."

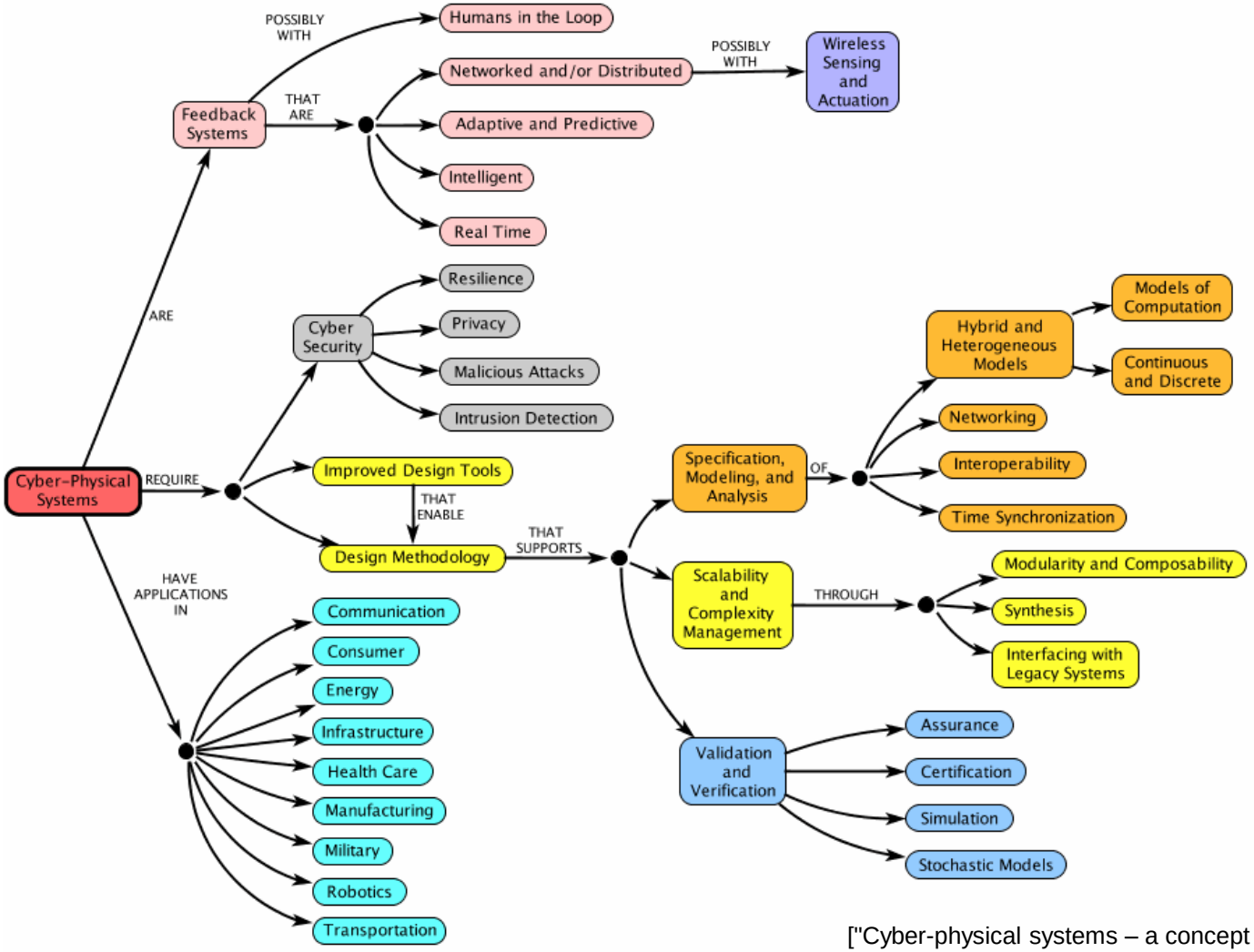
["High-Level System Modeling." Bergé et al. (1995)]

- *Typically*, ES are reactive systems
- Characteristics of reactive systems
  - Highly interactive (through sensors and actuators)
  - Nonterminating processes
  - Interrupt-driven
  - State-dependent response
  - Environment-oriented response
  - Parallel processes
  - Usually, stringent real-time requirements

["Design methods for reactive systems." R. J. Wieringa. (2003)]

Cyber-physical systems

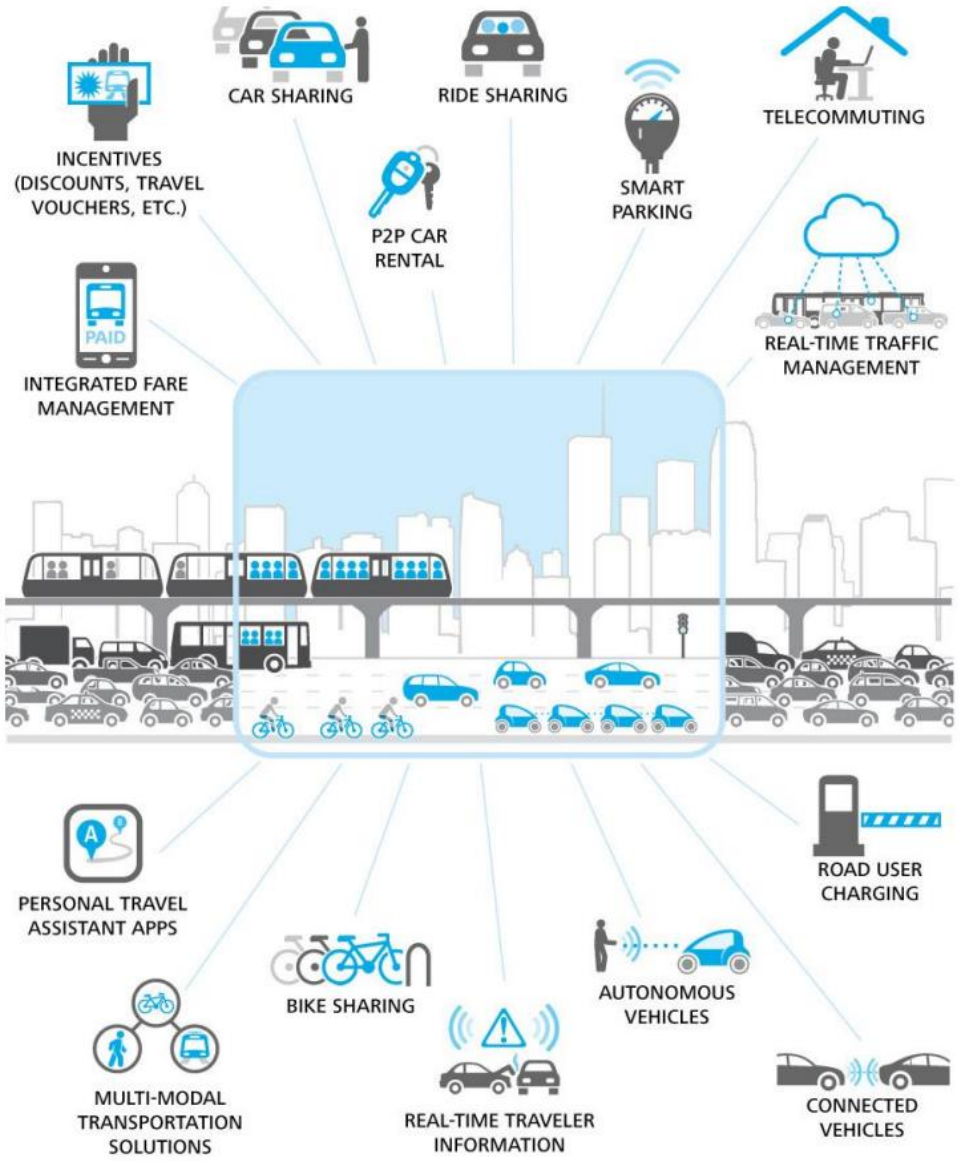
- US-American perspective: new word for embedded systems
- European perspective: *socio-technical systems that integrate IS, ES, and humans*



["Cyber-physical systems – a concept map." Sunder S.S et al.]

Cyber-physical systems

The TUM Living Lab Connected Mobility



["DigitalAge Transportation – The Future of Urban Mobility." Deloitte]

Information systems

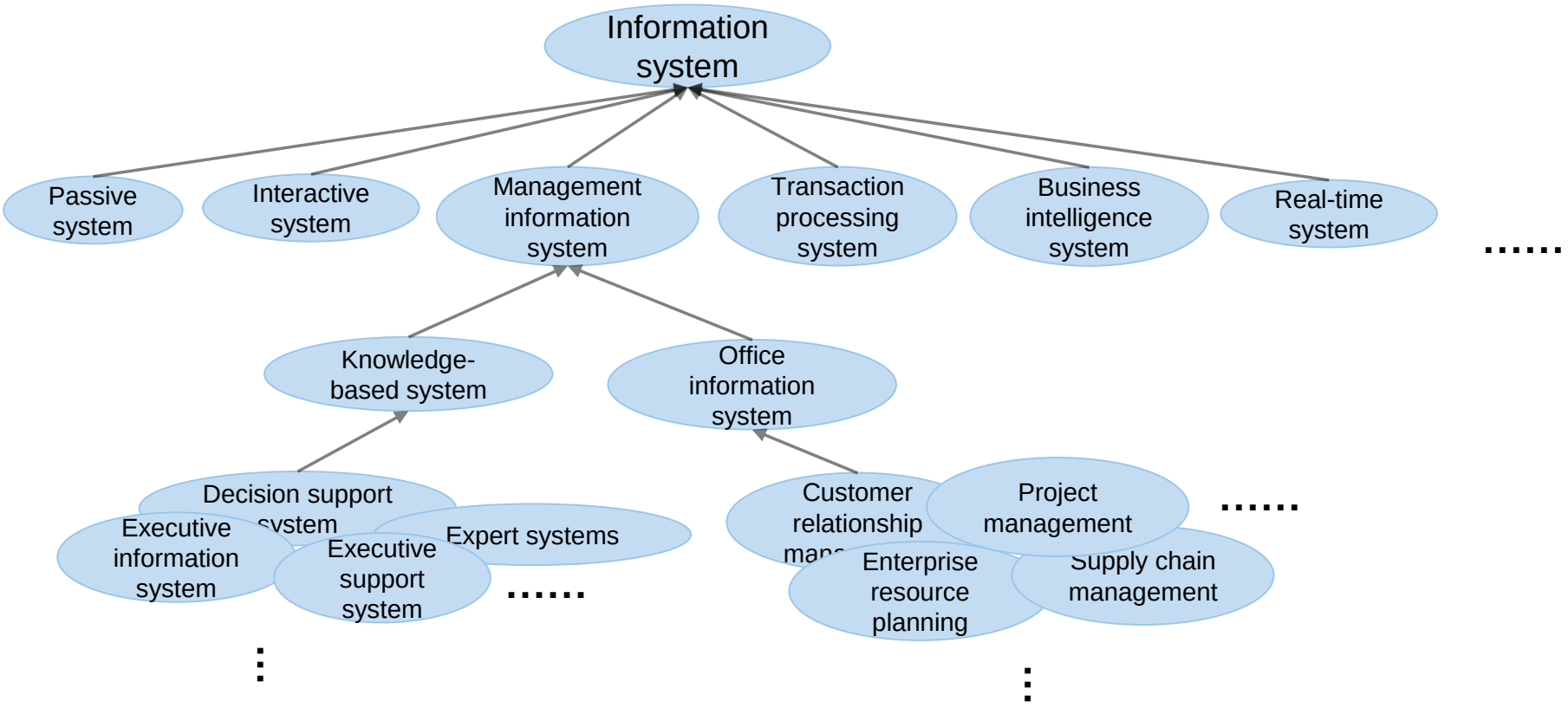
"An information system (IS) is an **integrated set of components for collecting, storing, and processing data and for delivering information, knowledge, and digital products**. Business firms and other organizations rely on information systems to carry out and manage their operations ..."

[Encyclopedia Britannica]

- Emphasis on software for supporting businesses
- In contrast to embedded systems, information systems have a more holistic view
  - combination of people, software, and hardware
- Data intensive systems
- Examples: ERP software, expert systems, …

Please note the difference between the above definition and the usage of the term "Informationssystem" in the information systems community!

Characteristics of information systems (1)



[Derived from "A Functional Taxonomy of Computer Based Information Systems." Mentzas G. (1994)]

Characteristics of information systems (2)

	Information system	Embedded system
Functionality	+	+
Usability	+	-/+
Reliability	+	+
Performance	+	+
Supportability	+	-/+
Availability	+	+
Scalability	+	-
Recoverability	+	-
Transaction support	+	-
Maintainability	+	-/+
Safety	-/+	+
Security	+	+
Code-size efficiency	-	+
Energy efficiency	-	+
Run-time efficiency	-/+	+
Physical constraints (weight)	-	+
Cost	+/-	+
Predictability	-	+
Deadline dependent	-	+
Quality of service	+	+

[FURPS Model. Robert Grady and Hewlett-Packard Co. (1987)]
[ISO/IEC 9126]