Advanced Topics of Software Engineering (ASE)

Chapter 1. The context of software engineering

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

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

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 is there 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	ECTS	Scope of the exam
IN2309	8	Everything that is covered in the lecture, exercises, and project

- **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 experience in at least one analysis and design technique, preferred UML
  - You are comfortable with reading documentations (e.g. setting up open source projects)

- **Beneficial:**
  - You have had practical experience with a large software system
  - You have already participated in a larger software project
  - You have experienced major software design problems

Organizational issues (2)

Lecture:

- Mondays: 12:15 - 13:45; (only online)
- Fridays: 08:30 - 10:00; (only online)

Central exercise:

- Mondays: 14:15 – 15:45; (only online)

Lecture and exercise material will be available on Moodle

- Registration for the course in TUM Online is required
- Additional registration for consulting hours groups in TUM Moodle
- Questions can be posted on the Moodle forum

Written examination

- Time and date: 25.02.2022
- **Must register for the exam on TUM Online separately**
- Language of examination – English only & on-site only
- Closed-book exam
  - English dictionary is allowed

Organizational issues (3)

Consulting Hours (Sprechstunden):

Group 1	Mondays	16:15 - 18:00	01.11.018	Tri
Group 2	Tuesday	10:15 - 12:00	01.11.018	Tri
Group 3	Tuesday	15:15 - 17:00	01.11.018	Tri
Group 4	Wednesday	12:15 - 14:00	01.11.018	Burak
Group 5	Wednesdays	14:15 - 16:00	01.10.011	Burak

Organizational issues (4)

Project bonus

- The accompanying project is optional, but **HIGHLY** recommended

- To get up to **1.0 grade bonus** for the final grade students have to
  - Pass the exam
  - Hand-in final executable project that satisfies the requirements
  - Short presentation of final project at the end of the semester
    - see “Project Specifications”

    → available on Moodle, discussed in central exercise session

- Not transferable to the next semesters
  - Applicable to **repetition** exam

What you will see

1. The context of software engineering

1.1. Introduction and overview

1.2. Characteristics of software systems

1.3. Factors affecting the design of a software system

1.4. Introduction to embedded systems (Guest lecture)

Structure of the lecture (1)

1. **The context of software engineering**
   - 1.1. Introduction and overview
   - 1.2. Characteristics of software systems
   - 1.3. Factors affecting the design of a software system
   - 1.4. Introduction to embedded systems (*Guest lecture*)

2. **From (quality) requirements to system design**
   - 2.1. Software architecture
   - 2.2. Antipatterns in software engineering
   - 2.3. Reuse
   - 2.4. Testability
   - 2.5. Safety
   - 2.6. Information security
   - 2.7. Introduction to Parallel Computing & Parallel Programming Models *(Guest Lecture)*

Make sure you attend the guest lectures, there will be exam questions from the talk!!

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
3.7. Blockchain-based 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
4.7. Software Architectures for High Performance Computing *(Guest Lecture)*

Learning objectives of the exercise

Objectives

- Recap and apply theoretical aspects from lecture to practical problems
- Hands-on state-of-the-art technology
- Develop a medium-sized hybrid IS-ES distributed system
- Simulation of typical junior developer work assignments

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 central exercises
  - Theoretical questions
  - Practical assignments

- Students work on their own, present solutions during exercises.

- Project
  - Students work in a group of 5.
  - **Optional** weekly consulting hours (Sprechstunden) for tutor feedback.
  - Final deliverables: Sources, system presentation, and report on individual contributions.

Structure of the exercise (2)

Weekly exercise assignments:

- Exercise sheets are published online every Monday, a week before their in-class realization.
- Students have a week to work on the solutions, and they present the solutions during the exercises.
- Exercise solutions will ***not*** be published.

Exercise structure:

1. Tutor recaps theory.
2. Tutor discusses theory questions interactively with students and explains requirements for practical assignments.
3. Students present solution of the theoretical questions and practical assignments.

Benefits of taking the theoretical exercises

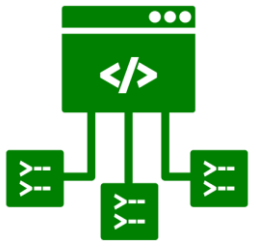
Getting familiar with the lecture content

Practicing software engineering concepts

Working with tools used in the industry

Preparing for the final exam

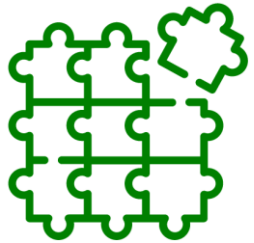
Benefits of taking the practical exercises



Experience with popular frameworks

Learn theory from practice

Solve challenges in SE



Explore various approaches in SE

Beneficial for the Final Exam

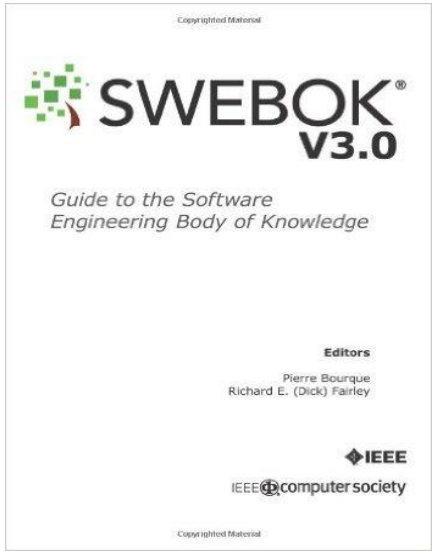
Practice, framework, challenge, puzzle by Sumit Saengthong, Shocho, Adrien Coquet, coloripop from the Noun Project

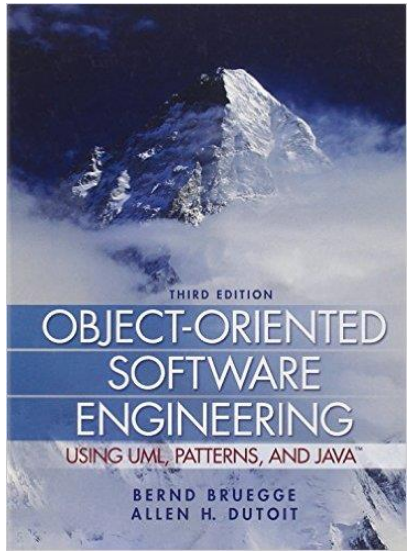
Out of scope for this lecture

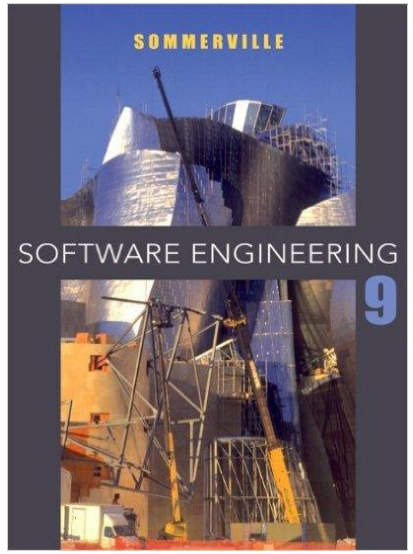
We do not cover the following topics in depth

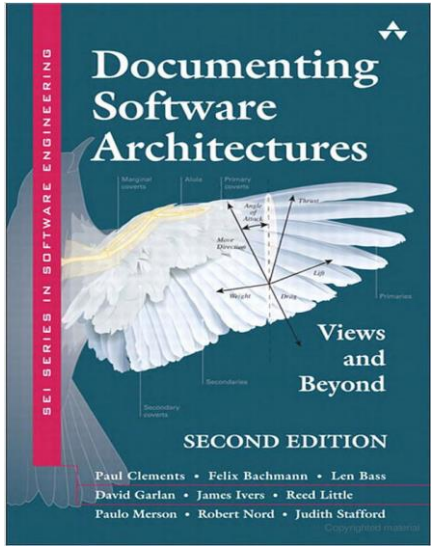
▪ Software engineering processes	Introduction to software engineering (IN0006)
▪ Requirements engineering	Requirements engineering (IN2198)
▪ Project management	Project organization and management (IN2083)
▪ Design patterns	Patterns in software engineering (IN2081)
▪ Software testing	Advanced topics 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)
▪ Functional safety and information security	Funktionale Sicherheit (IN 2247), Security engineering (IN 2178), Secure coding (IN 2106)
▪ Topics related to Blockchain	Blockchain-based systems engineering (IN2359)

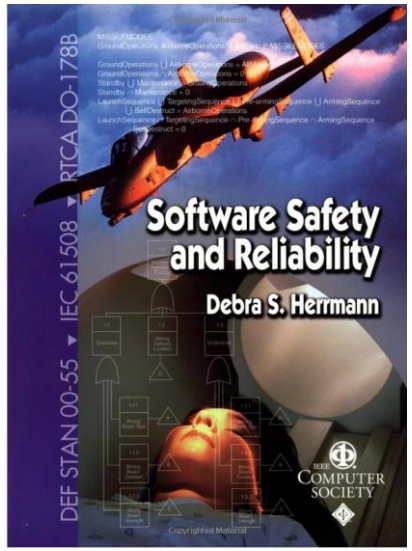
Reference books

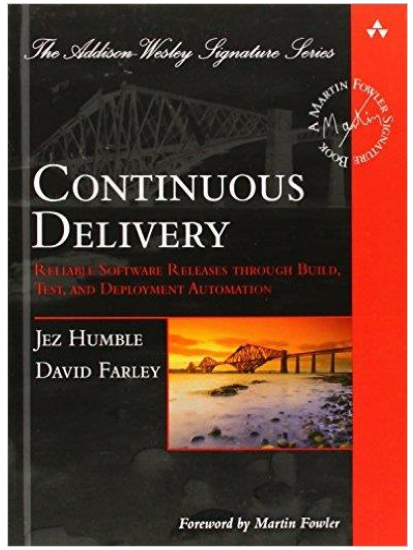












Additional references

- "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).
- “Foundations of Software Testing”, 2nd ed., Mathur A. (2013).
- "Softwaretechnik: Praxiswissen für Software-Ingenieure" Siedersleben, J. (2003).
- "Pro Git“ Chacon S., Straub B. (2014).

1. The context of software engineering

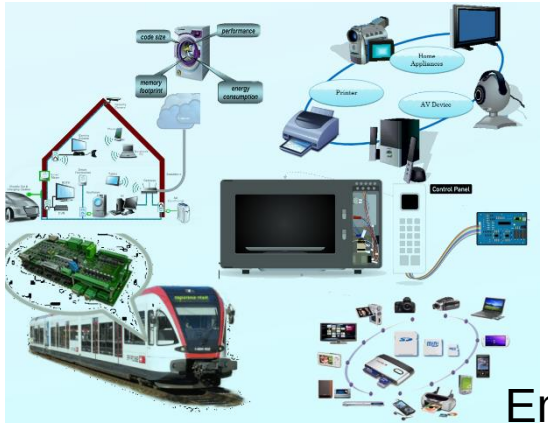
1.1. Introduction and overview

1.2. Characteristics of software systems

1.3. Factors affecting the design of a software system

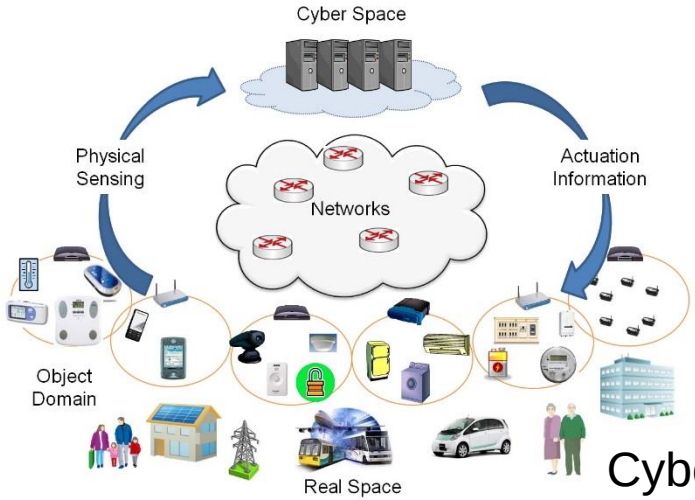
1.4. Introduction to embedded systems (Guest lecture)

Application domains



Embedded systems

Scientific software systems



Cyber-physical systems

Information systems

1. The context of software engineering

1.1. Introduction and overview

1.2. Characteristics of software systems

1.2.1. Embedded systems

1.2.2. Information systems

1.2.3. Cyber-physical systems

1.3. Introduction to embedded systems (Guest lecture)

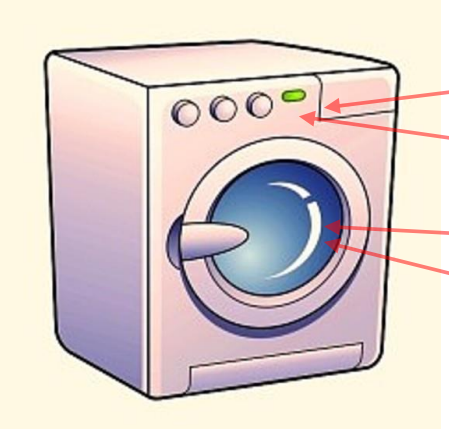
1.4. Factors affecting the design of a software system

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

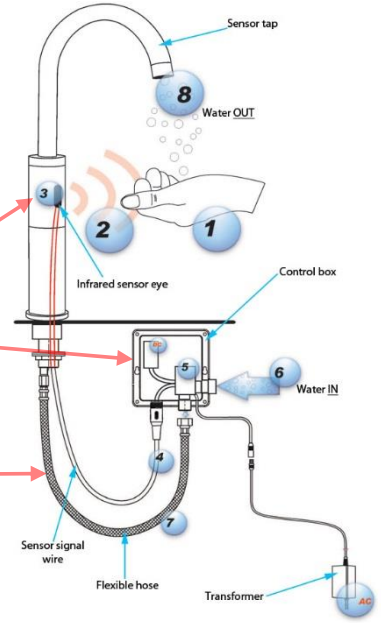


User interface

Processor

Sensors

Actuators



Characteristics of embedded systems (1)

Processing: ability to process the analog/digital signals

- Intricate control flows in control loops
- Open-loop and closed-loop
- Discrete and continuous

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 (3)

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

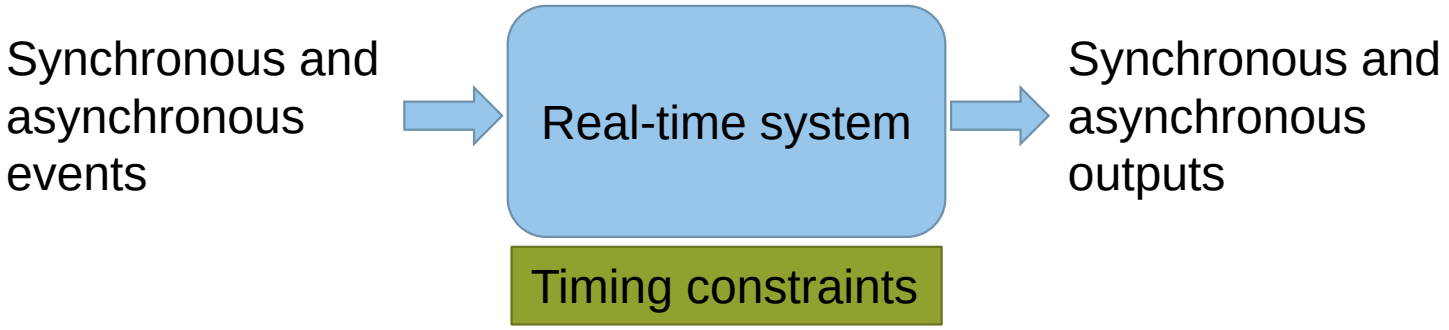
Dependability

Efficiency

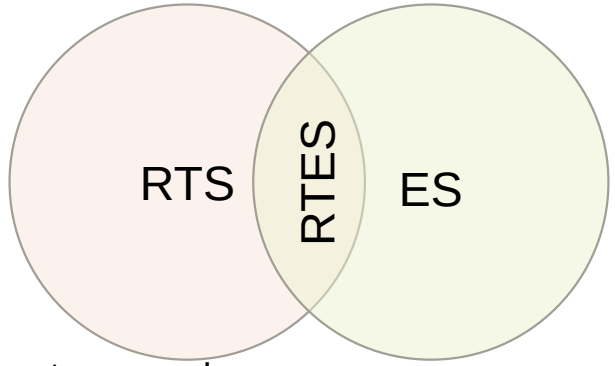
Real-time

["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)
  - Non-terminating 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)]

1. The context of software engineering

1.1. Introduction and overview

1.2. Characteristics of software systems

1.2.1. Embedded systems

1.2.2. Information systems

1.2.3. Cyber-physical systems

1.3. Factors affecting the design of a software system

1.4. Introduction to embedded systems (Guest lecture)

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]

- In contrast to embedded systems, information systems have a more holistic view
  - combination of people, software, and hardware

- Emphasis on software for supporting businesses and organizations
- Data-intensive systems
- Examples: ERP software, e-commerce systems, supply chain management, …

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

Evolution of information systems (1)

1950s – 1960s	Electronic data processing systems: the first business application of computers performed repetitive, high-volume, **transaction-computing tasks**. The computers “crunched numbers” summarizing and organizing transactions and data in the accounting, finance, and human resources areas. Such systems are generally called transaction processing systems (TPSs).
1960s – 1970s	Management Information Systems (MISs): these systems access, organize, summarize and display information for supporting routine **decision making** in **functional areas**.
1970s – 1980s	Decision Support Systems: were developed to provide computer-based support for complex, non-routine **managerial decision-making process**.

[“Introduction to Information Technology.” John Wiley & Sons, Inc. (2005)]

Evolution of information systems (2)

1980s – mid- 1990s

End-user computing: The use or development of information systems by the principal users of the systems' outputs, such as analysts, managers, and other professionals.

Office Automation Systems (OASs): such as **word processing systems** were developed to support office and clerical workers.

Intelligent Support System (ISSs): Include **expert systems** which provide the stored knowledge of experts to non-experts, and a new type of intelligent systems with **machine-learning capabilities that can learn from historical cases**.

Knowledge Management Systems: Support the creating, gathering, organizing, integrating and **disseminating of an organization knowledge**.

Data Warehousing: A data warehouse is a database designed to support DSS, ISS, and other analytical and end-user activities.

["Introduction to Information Technology." John Wiley & Sons, Inc. (2005)]

Evolution of information systems (3)

late 1990s

Enterprise resource planning (ERP) systems: organization-specific form of a strategic IS that **integrates all facets of a firm**, including its planning, manufacturing, sales, resource management, customer relations, inventory control, order tracking, financial management, human resources and marketing – virtually every business function.

Mobile computing: IS that supports employees who are working with customers or business partners outside the physical boundaries of their companies; can be done over wire line or wireless networks.

since 2000

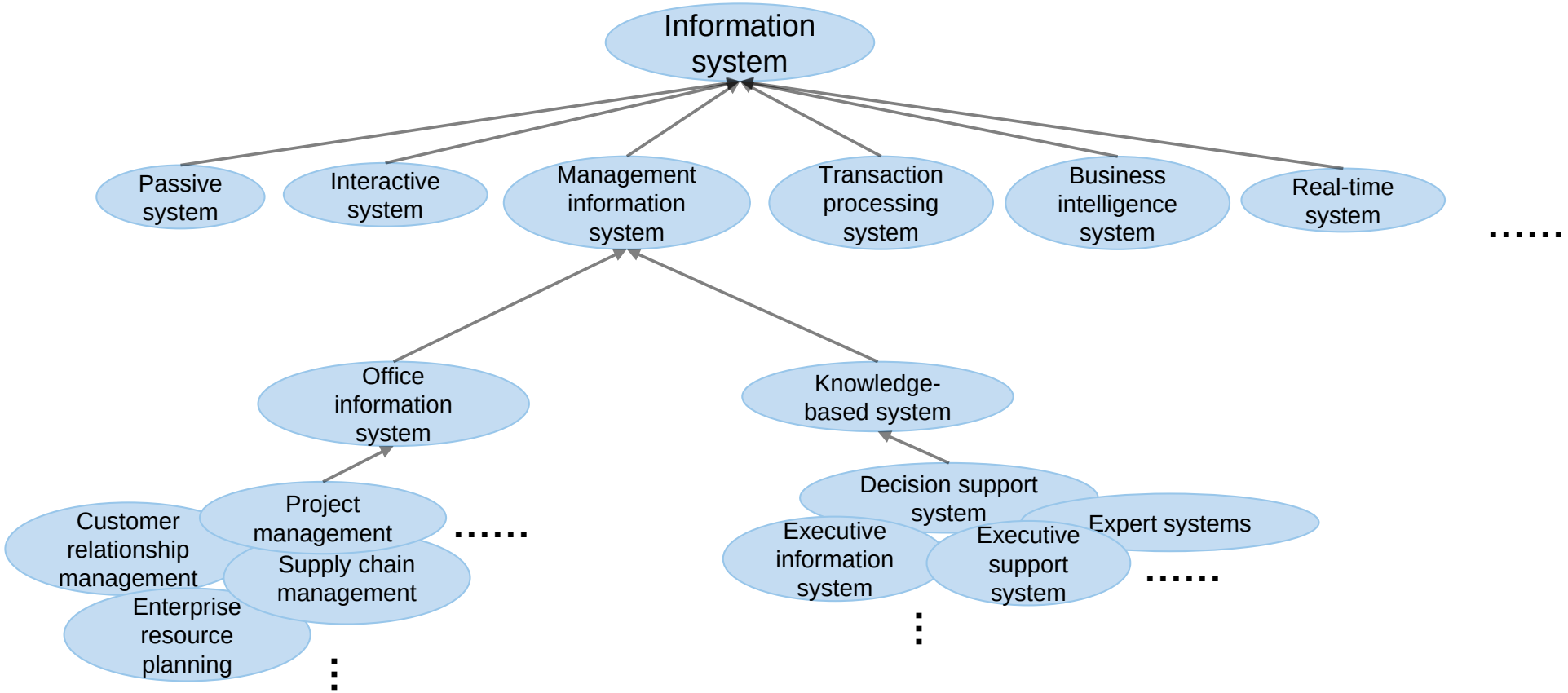
Internet-based **e-business and e-commerce**: Web-enabled enterprises and global e-business operations and e-commerce on the internet, intranets, extranets, and other networks.

since 2010

Big data, Cloud computing, Platform ecosystems, Blockchain-based systems, ...

["Introduction to Information Technology." John Wiley & Sons, Inc. (2005)]

Types of information systems



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

Characteristics of information systems (2)

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

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

1. The context of software engineering

1.1. Introduction and overview

1.2. Characteristics of software systems

1.2.1. Embedded systems

1.2.2. Information systems

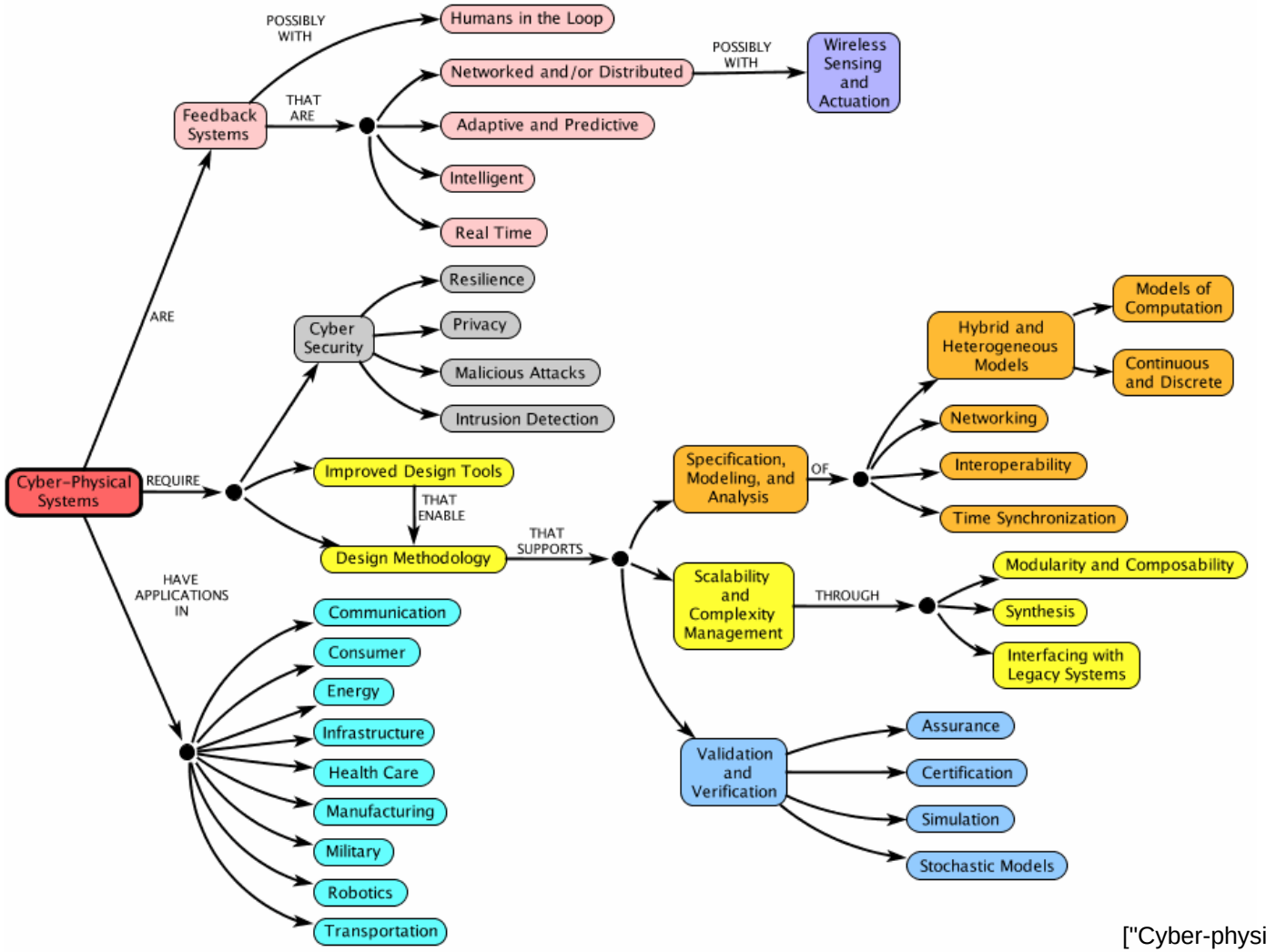
1.2.3. Cyber-physical systems

1.3. Factors affecting the design of a software system

1.4. Introduction to embedded systems (Guest lecture)

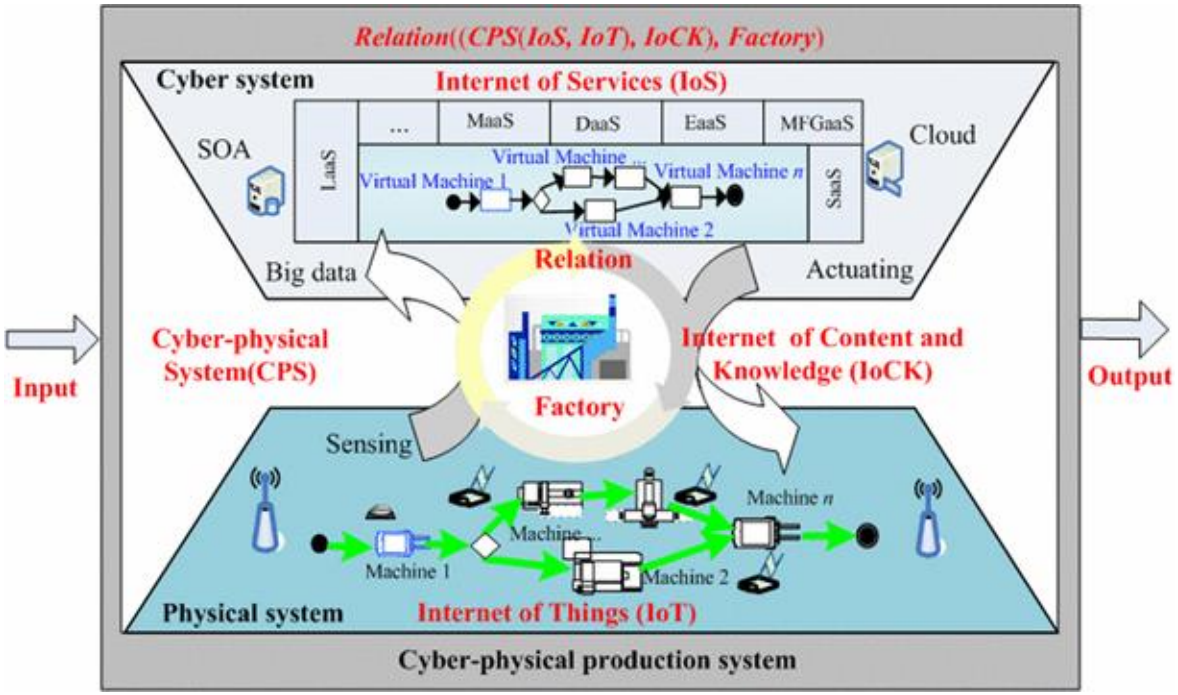
Cyber-physical systems

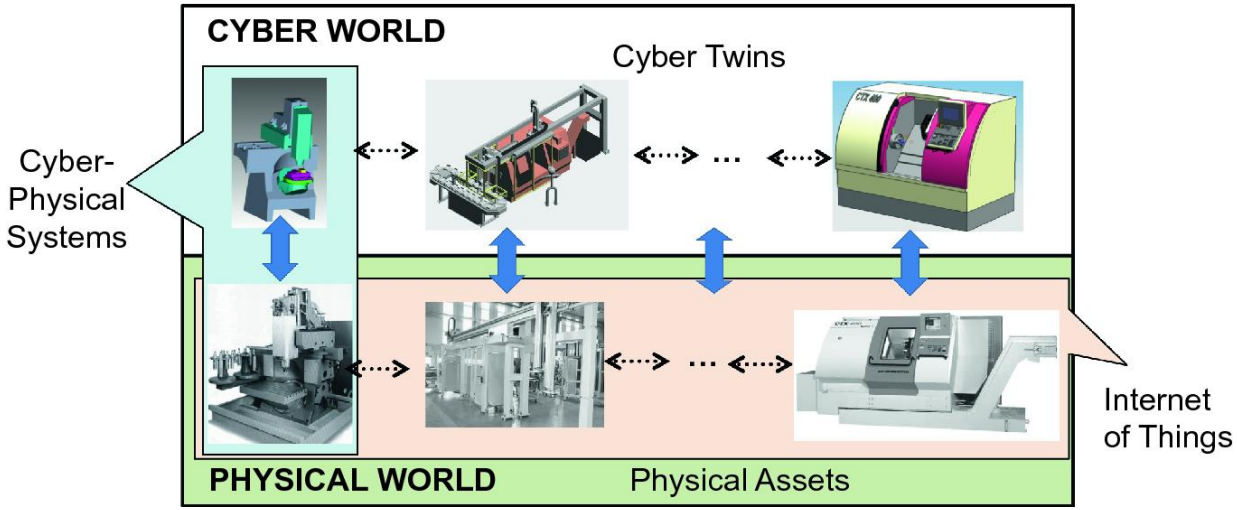
- US-American perspective: new word for embedded systems
- European perspective: *socio-technical systems that integrate IS. ES. and humans*



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

Cyber-physical systems and IoT



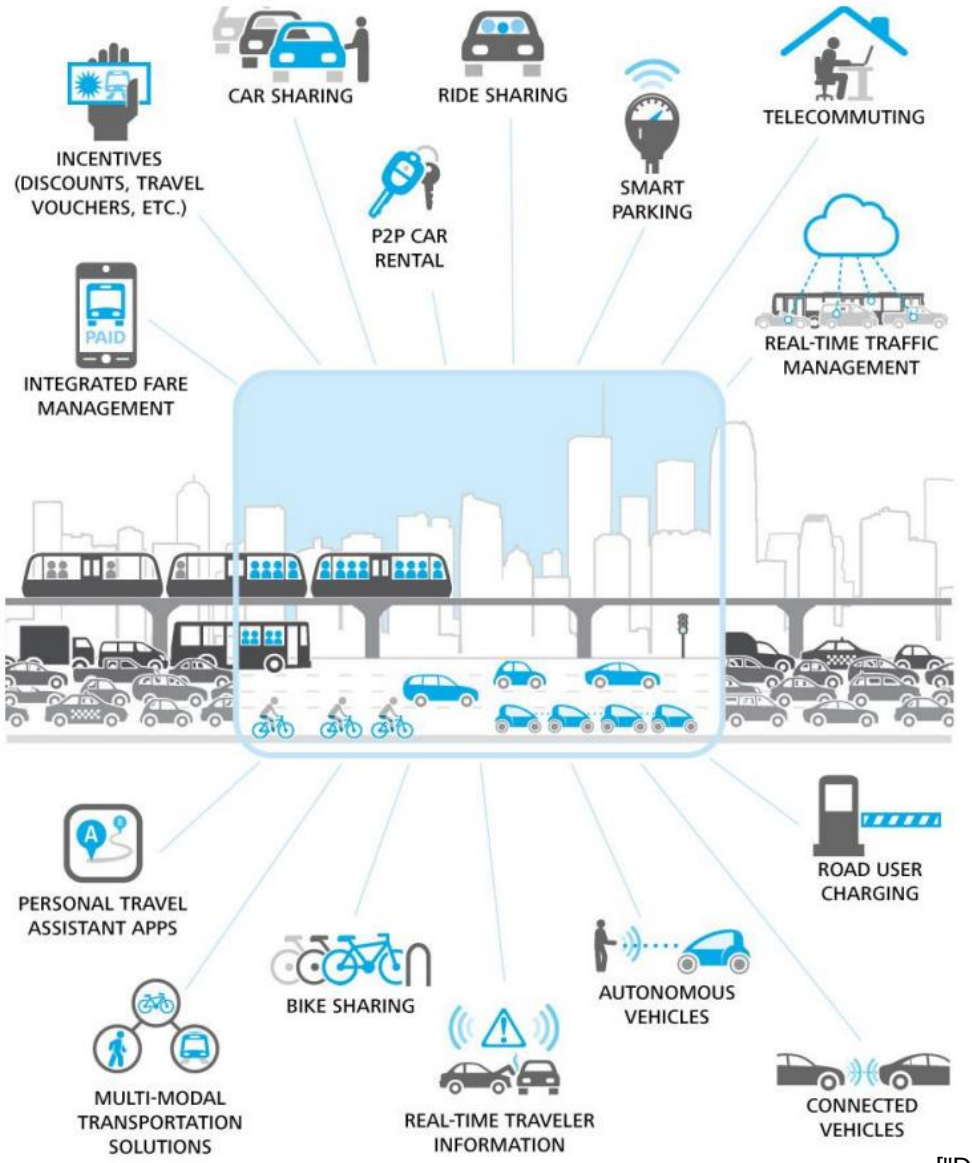


[„Smart manufacturing based on cyber-physical systems and beyond “ Yao X. et al]

[„Mechatronic and Cyber-Physical Systems within the Domain of the Internet of Things“Hehenberger P. et al]

Cyber-physical systems

The TUM Living Lab Connected Mobility



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

1. The context of software engineering

1.1. Introduction and overview

1.2. Characteristics of software systems

1.3. Factors affecting the design of a software system

1.3.1. Processes in software engineering

1.3.2. Artifacts in software engineering

1.3.3. Quality

1.4. Introduction to embedded systems (Guest lecture)

This is a recap.
These topics have extensively been 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)]

What is software engineering? (3)

Pretschner

- To me, it’s understanding trade-offs, making related informed decisions, and implementing them.
- Structured, reproducible approach.
- Scientific principles.
- Customer value.

Matthes

- For me, it‘s understanding people and organizations, and their relationships.
- Understanding trade-offs, making related informed decisions, and implementing them.
- Engineering principles.

Techniques, Methodologies and Tools

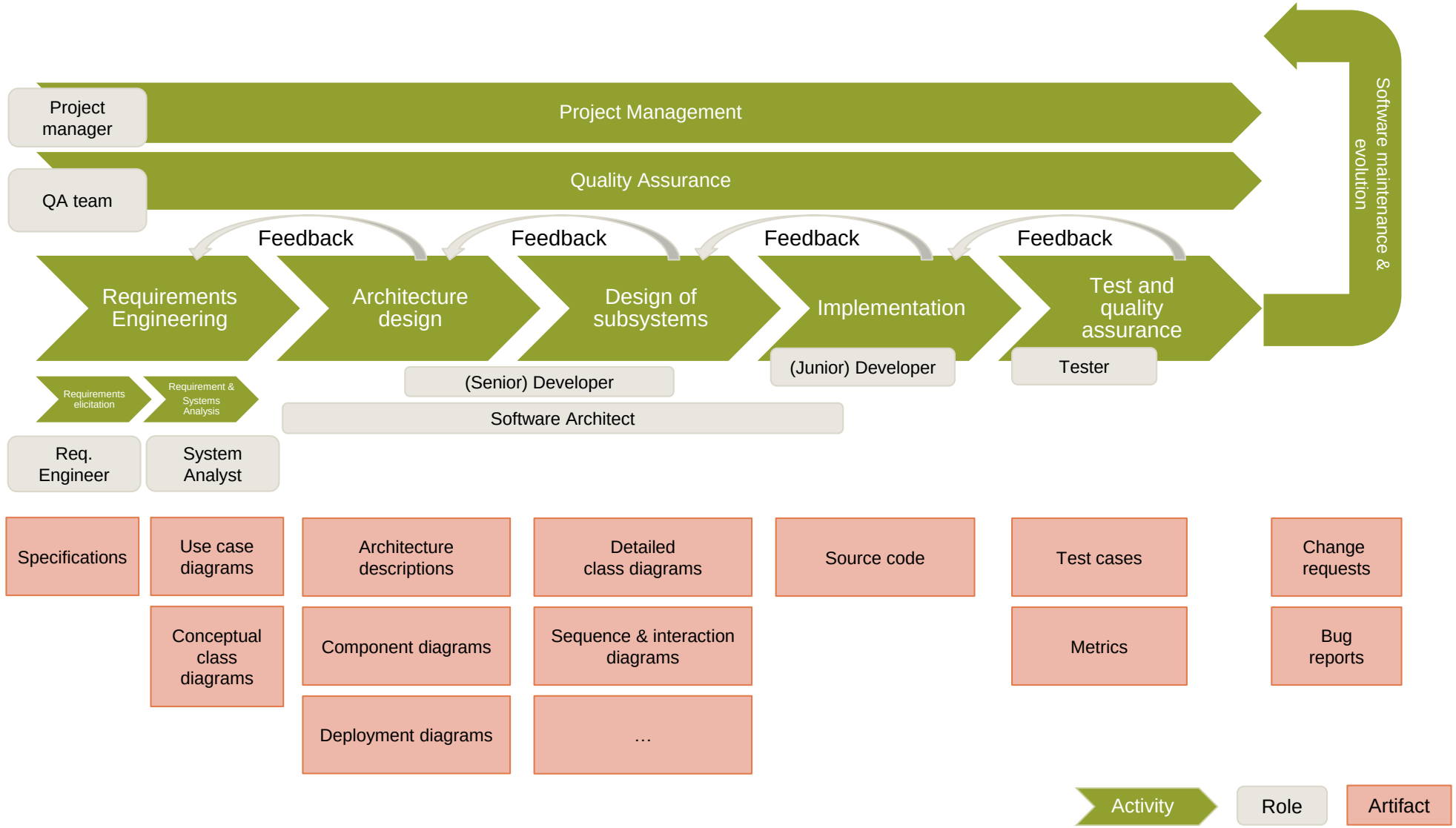
- **Techniques:**
  - Formal procedures for producing results using some well-defined notation

- **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

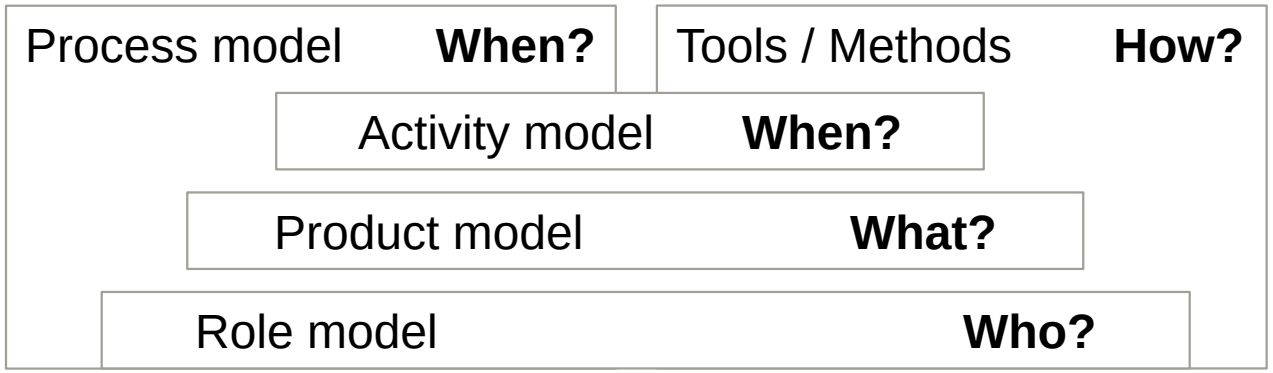


The process model

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:

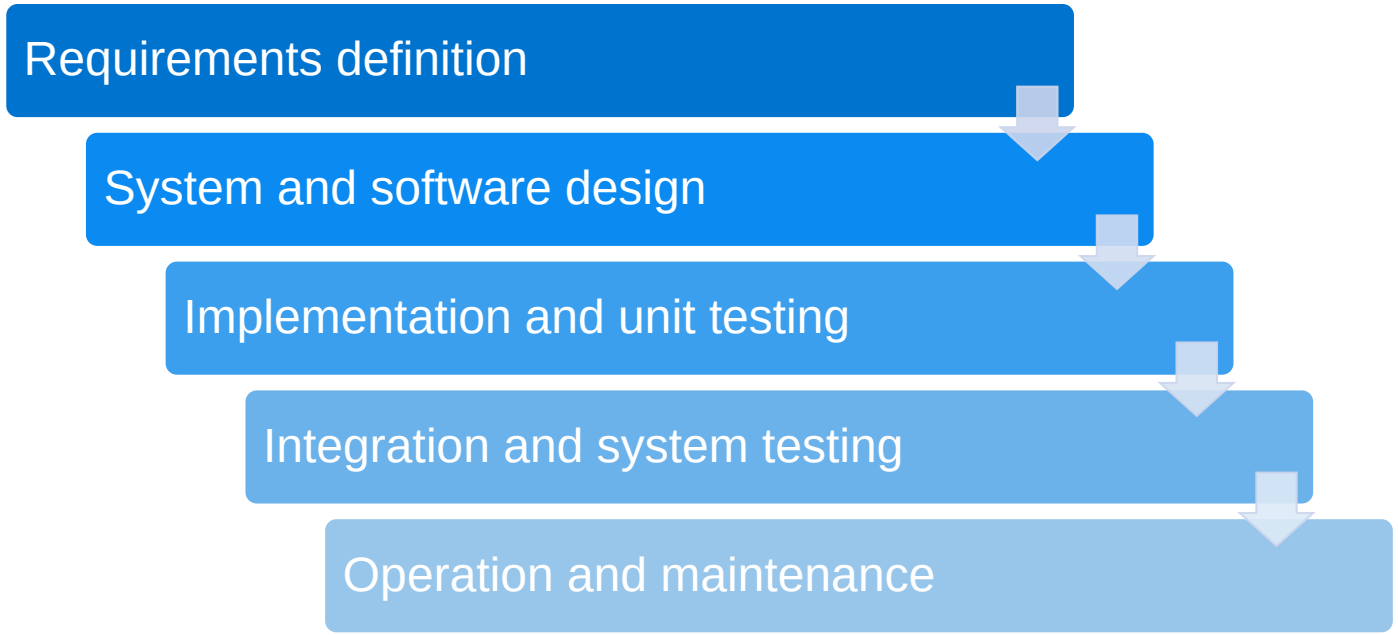
- Waterfall model (phase-oriented, sequential)
- Iterative model (phase-oriented, multiple pass)
- Incremental model (development in expansion stages)

Areas of conflict

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

The waterfall model

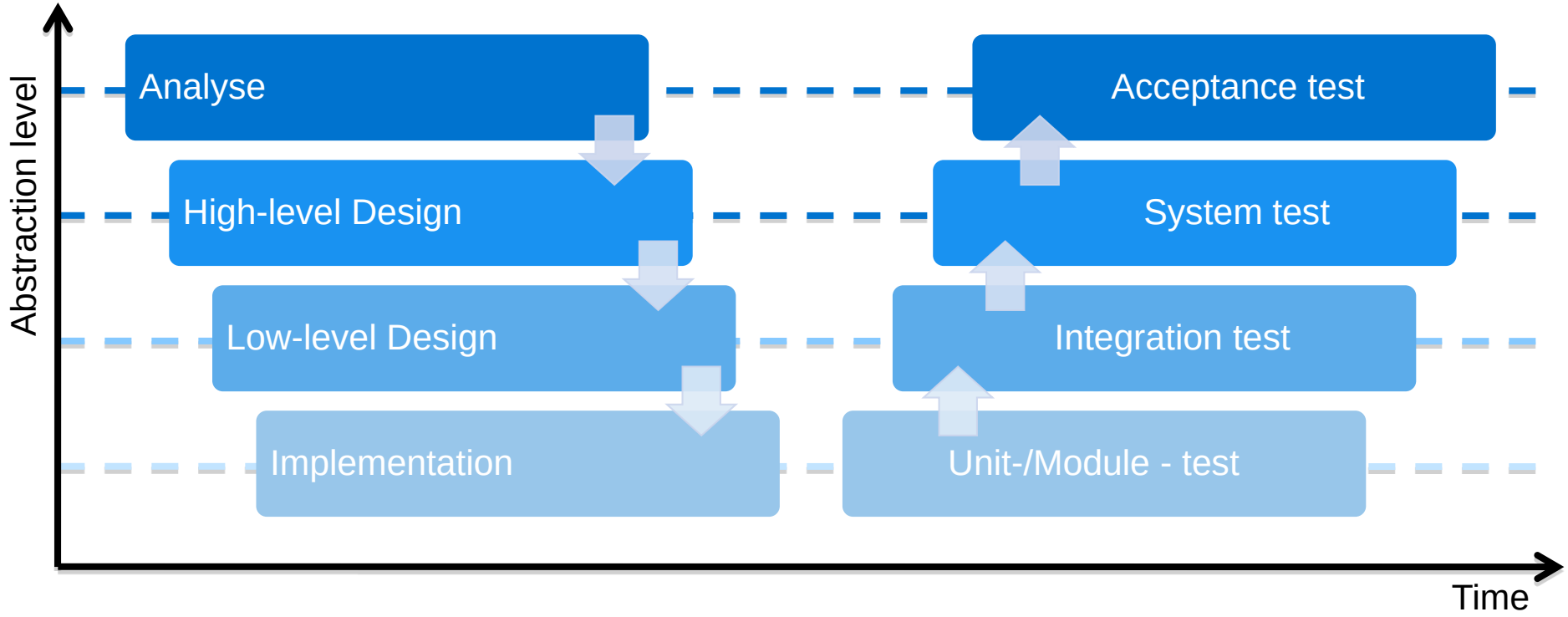
- Classical / conventional process model (goes back to Royce, 1970)
- Inflexible (sequential procedure is enforced)
- Risky (errors are found late and are expensive)



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

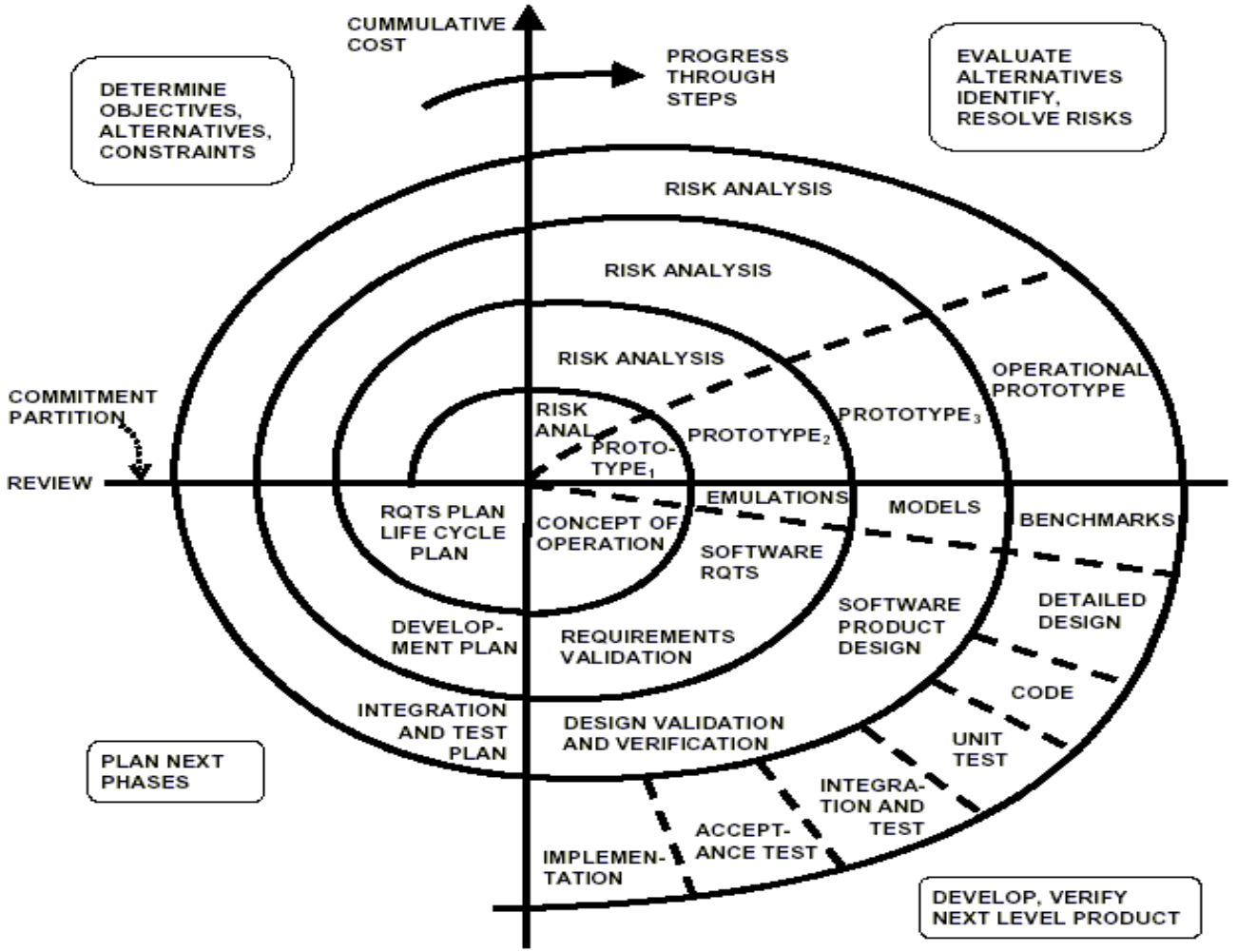
The spiral model

These include:

- Incremental
- Iterative
- Evolutionary models

- Self-learning process

- Aim: *risk minimization*

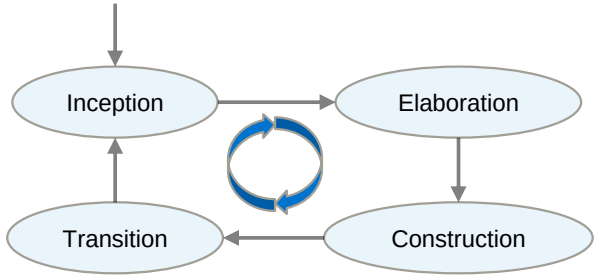


["A spiral model of software development and enhancement." Boehm B.W. (1988)]

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

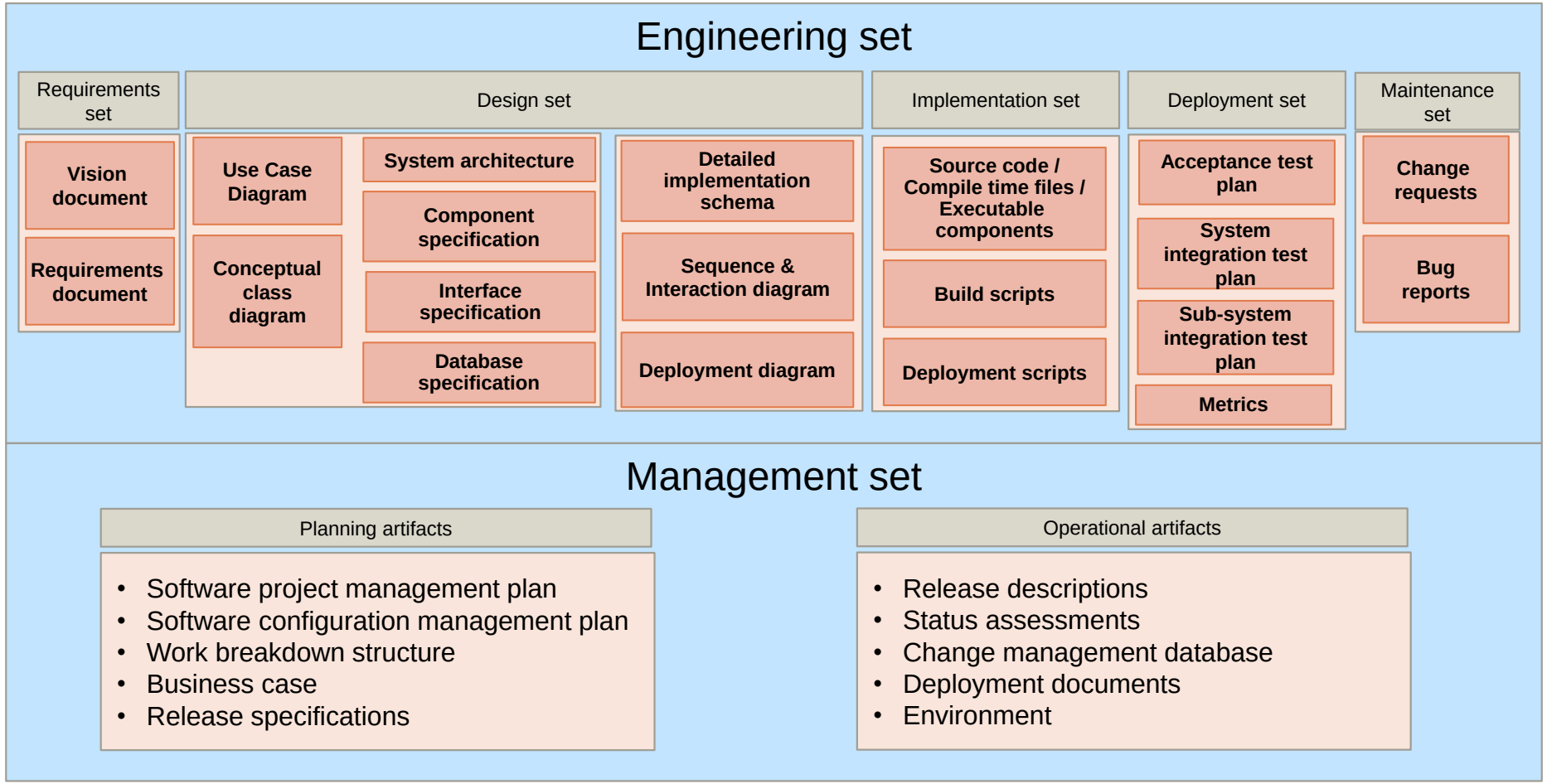
- **Phase:** Status of the software system.
  - 4 phases: Inception, Elaboration, Construction, Transition
- **Workflow:** Mostly sequential activity that produces artifacts
  - 7 workflows: Management, environment, requirements, design, implementation, assessment, deployment
  - *5 artifact sets: Management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

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.



Agile development

The aim is a leaner, less bureaucratic development process with

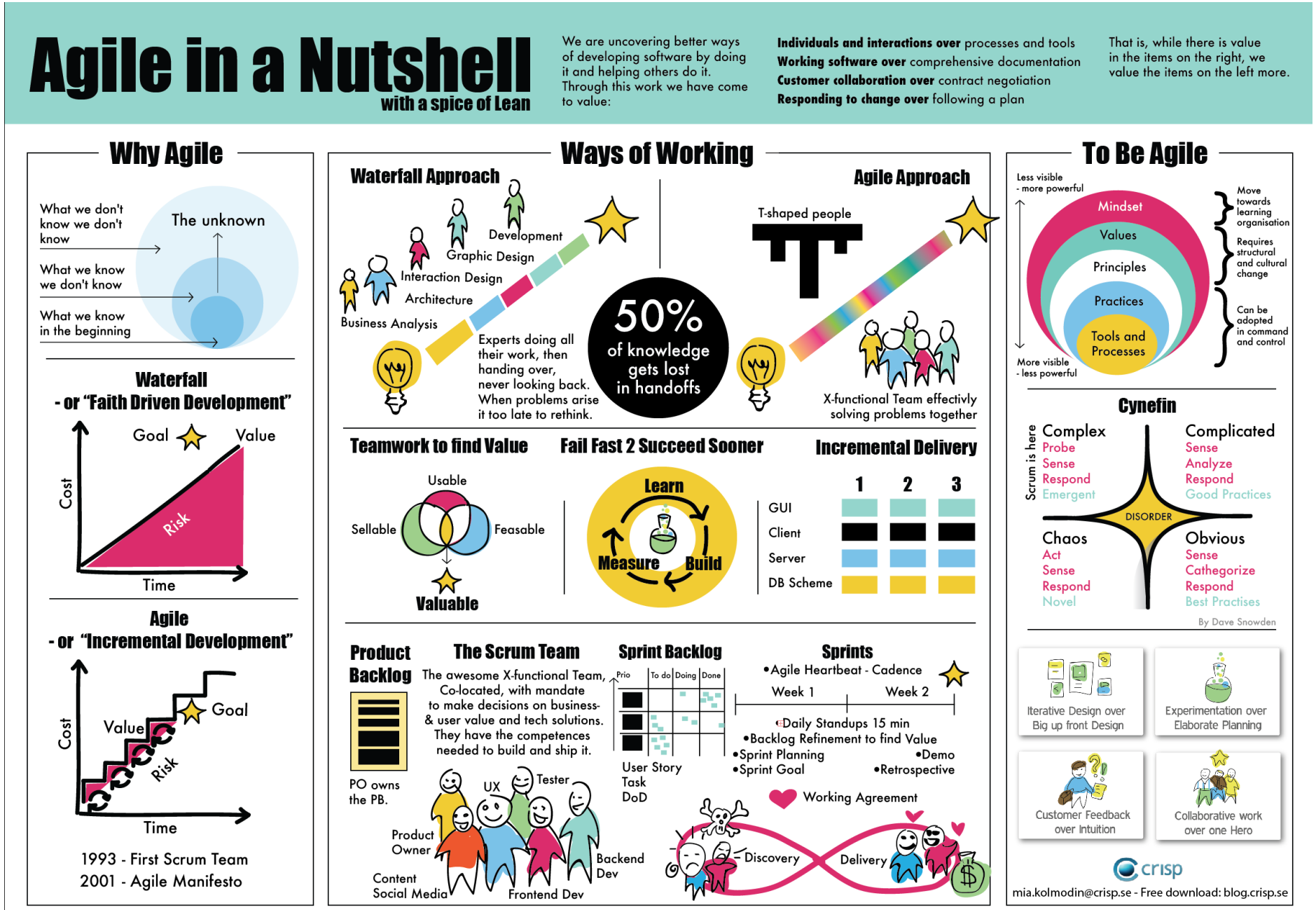
- fewer rules and
- a focus on working software that solves the customer's problem.

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.

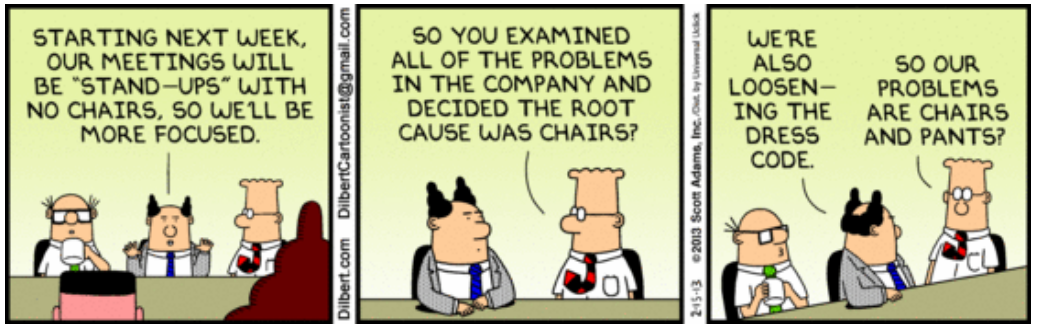
[from the Agile Manifesto - www.agilemanifesto.org]



https://image-store.slidesharecdn.com/47a0f518-18fa-4a03-b6e5-91d2c4379381-original.png

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.

Agility is more than a software engineering process model

- Fundamental consequences on the role of (intended) software architecture (as a planning document)
- Fundamental consequences on the role of specifications (as a planning document)
- Fundamental consequences on the role of leadership, specialists, hierarchy in a team, etc.

Main challenges:

- Inflexibility of legacy systems
- Inflexibility of of-the-shelf software
- Scaling to very large products
- Scaling to multiple products in an application portfolio
- Integration with outsourcing approaches
- Incompatibility with established (fixed-priced) tendering processes

There are promising developments in practice and research to solve these challenges
(see lecture “Introduction to scaled agile software development”)

1. The context of software engineering

1.1. Introduction and overview

1.2. Characteristics of software systems

1.3. Factors affecting the design of a software system

1.3.1. Processes in software engineering

1.3.2. Quality

1.4. Introduction to embedded systems (Guest lecture)

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."

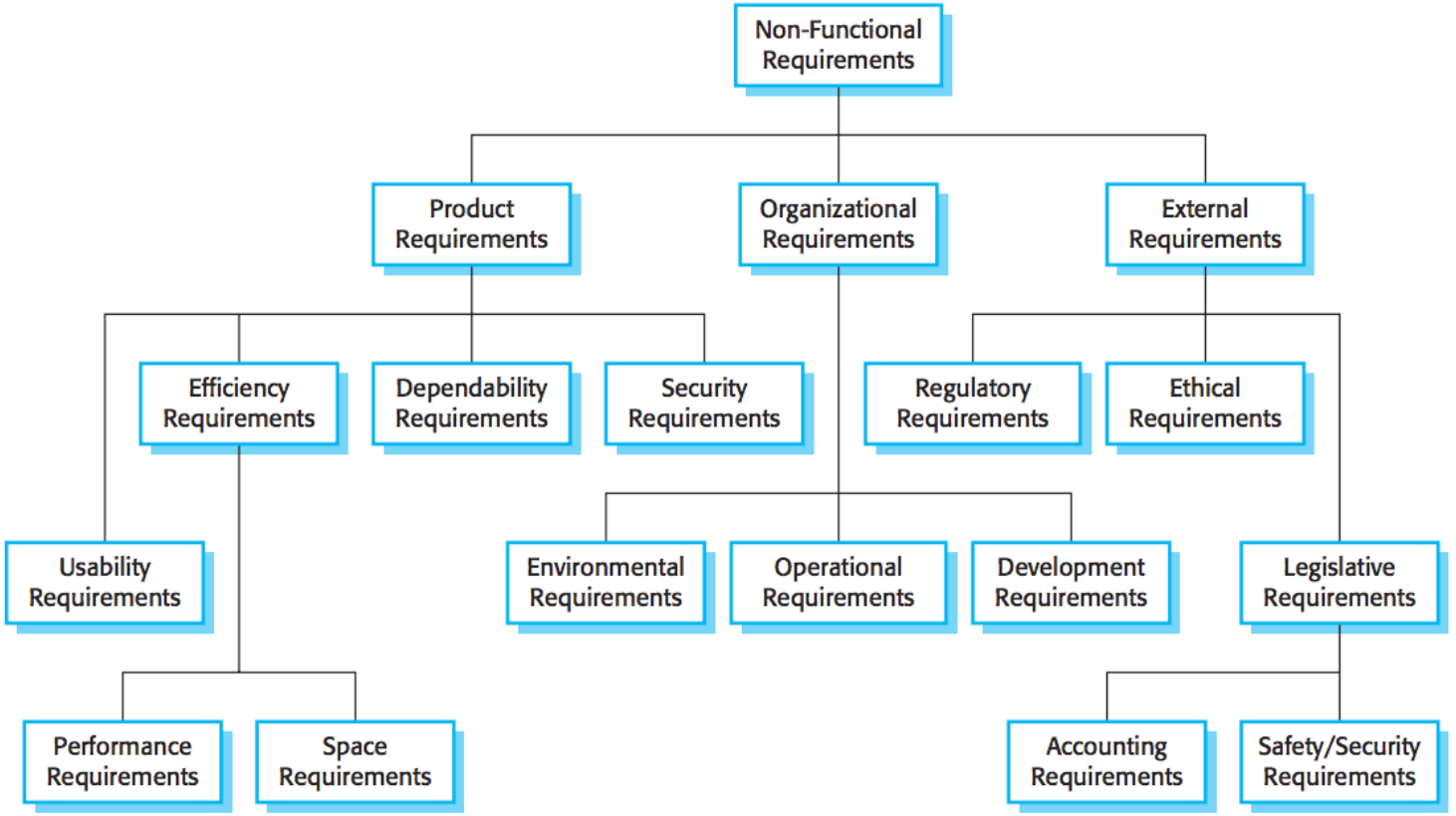
["ISO 9000 - Quality management" http://www.iso.org/iso/iso_9000]

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

Software quality

- Quality is the key factor for the product's success and customers' satisfaction.
- “Quality” is multi-dimensional!
- 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, CMMI, SPICE, ITIL, …
  - Rationale: Mature processes lead to good products … always true?
  - Quality also needs to be addressed at the product level!

Quality and non-functional requirements



["Software Engineering." Ian Sommerville. (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**
  - According to ISO 9126 *functionality* is a quality aspect but typically not a non-functional requirement

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

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

Quality and trade-offs

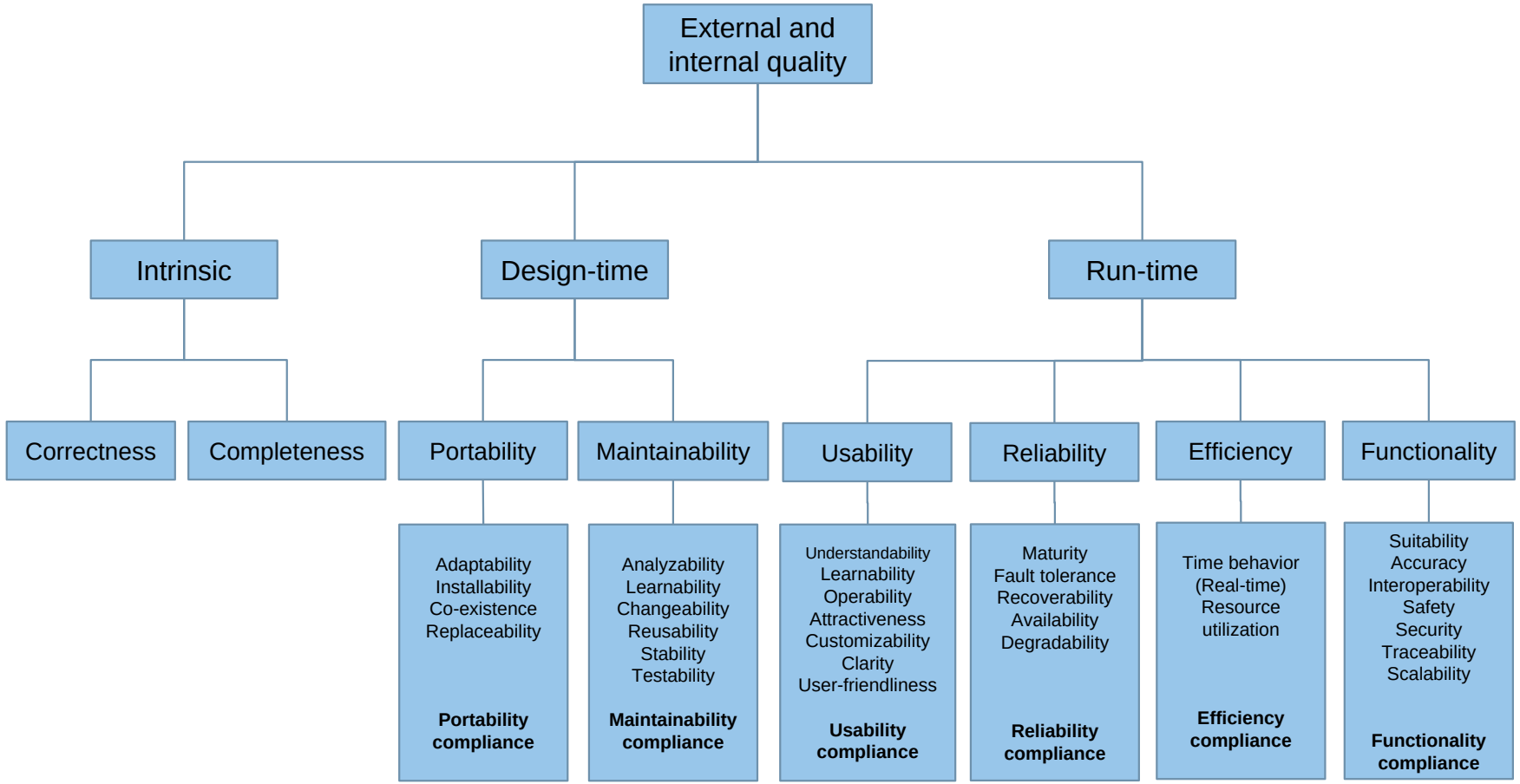
- Typically, there exist interdependencies between quality aspects
  - Positive: one quality aspect supports the other
    (e.g. maintainability vs. portability)
  - Negative: one quality aspect interferes with the other
    (e.g. security vs. efficiency)

- 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

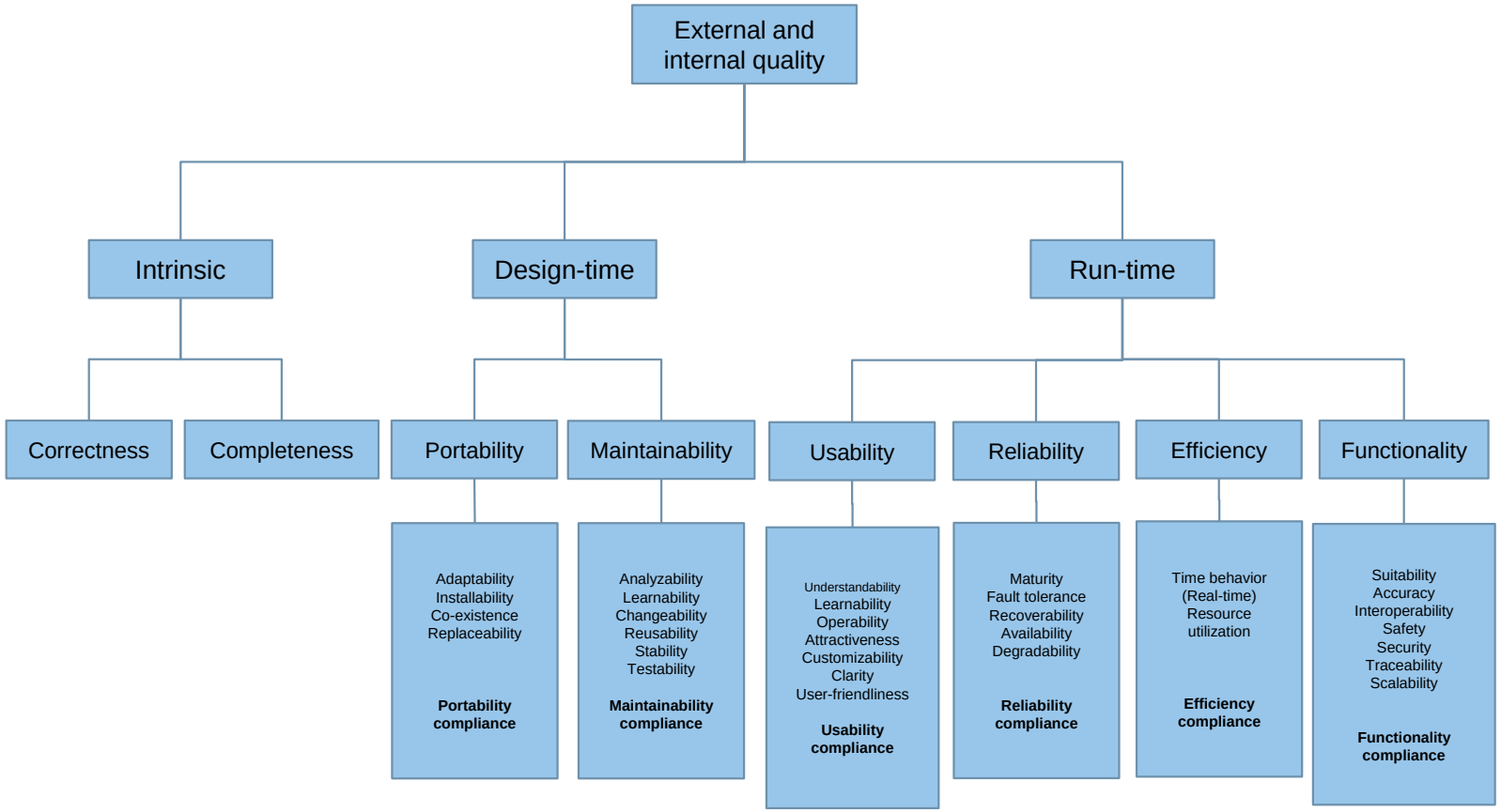
- (Diverse aspects of) Quality fundamentally influenced by a system's architecture!

Types of quality



[ISO/IEC 9126]

Types of quality

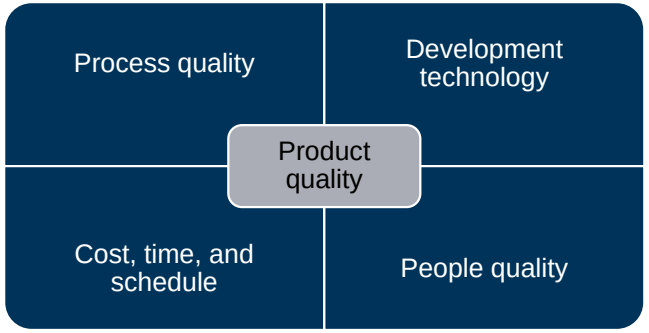


- These attributes refer both to the external as well as internal quality attributes.
- The quality attributes are applicable at multiple levels of granularity. For instance, one can define these attributes from a black-box or white-box perspective.

[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

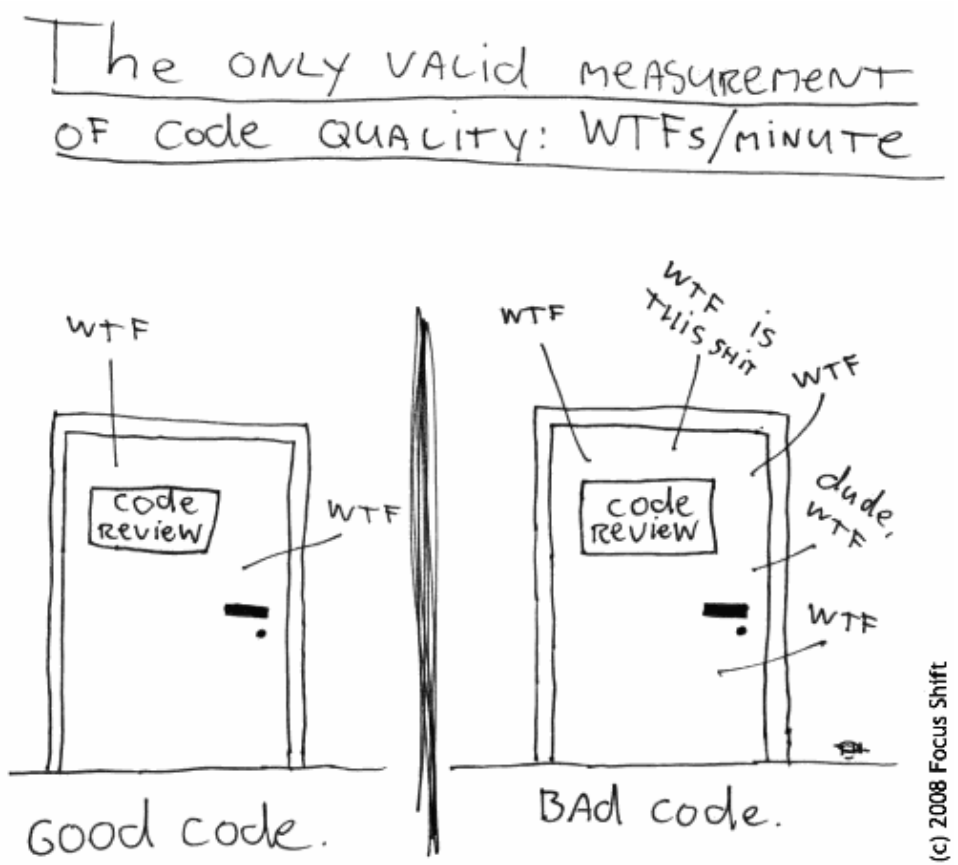


- Software is not manufactured but designed
- Software development is a creative rather than a mechanical process

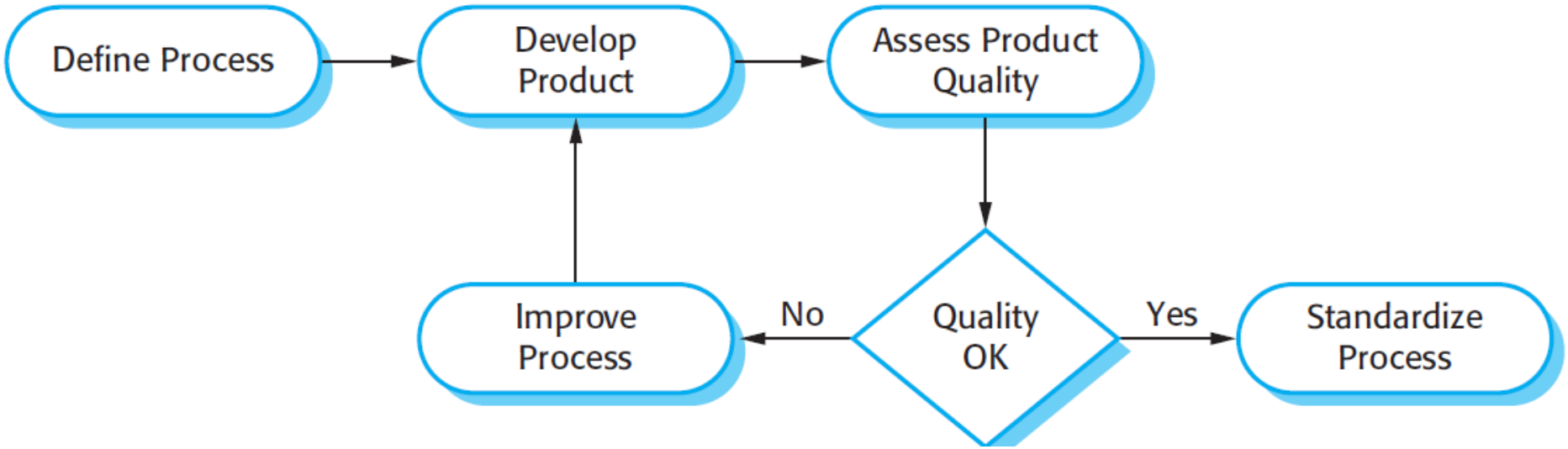
["Software Engineering." Ian Sommerville. (2010)]

Process and product quality attributes (2)

- It is difficult to measure software quality attributes directly
  - E.g., maintainability cannot be measured without using the software for a long time
  - Product metrics try to capture quality attributes



Process-based quality



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

Quality assurance and standards

- Standards are one 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.
- Standards help others to better estimate product quality (hopefully).

- 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 Sommerville. (2010)]

Product and process standards

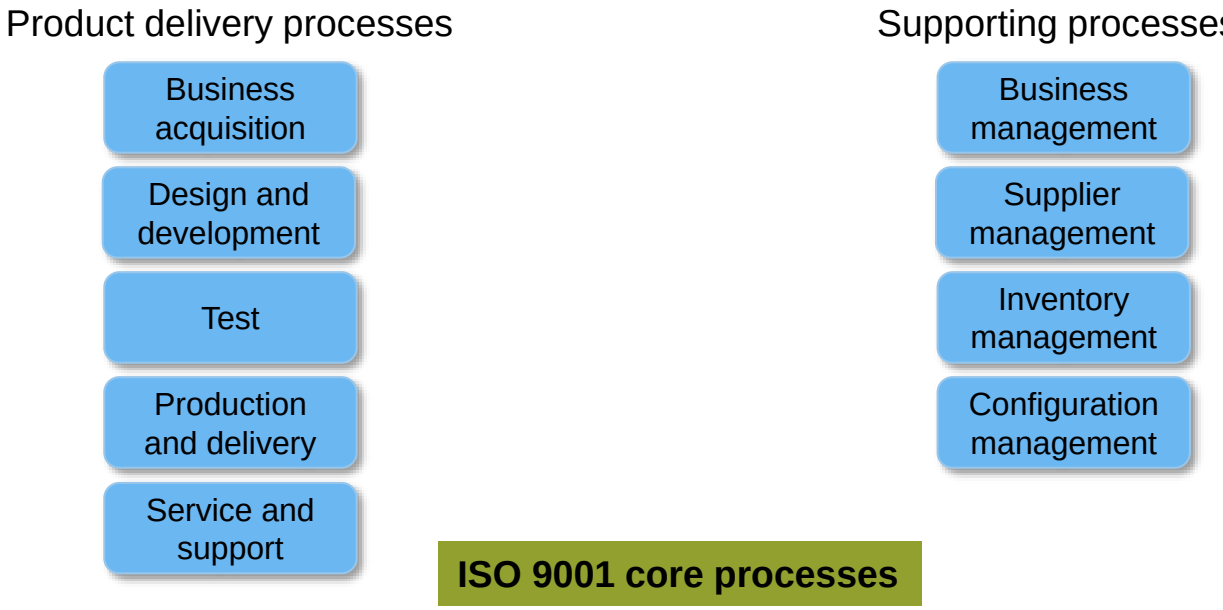
Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of code for system building
Procedure header format	Version release process
Programming style and conventions	Project plan approval process
Project plan format	Change control process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

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

ISO 9000 and ISO 9001

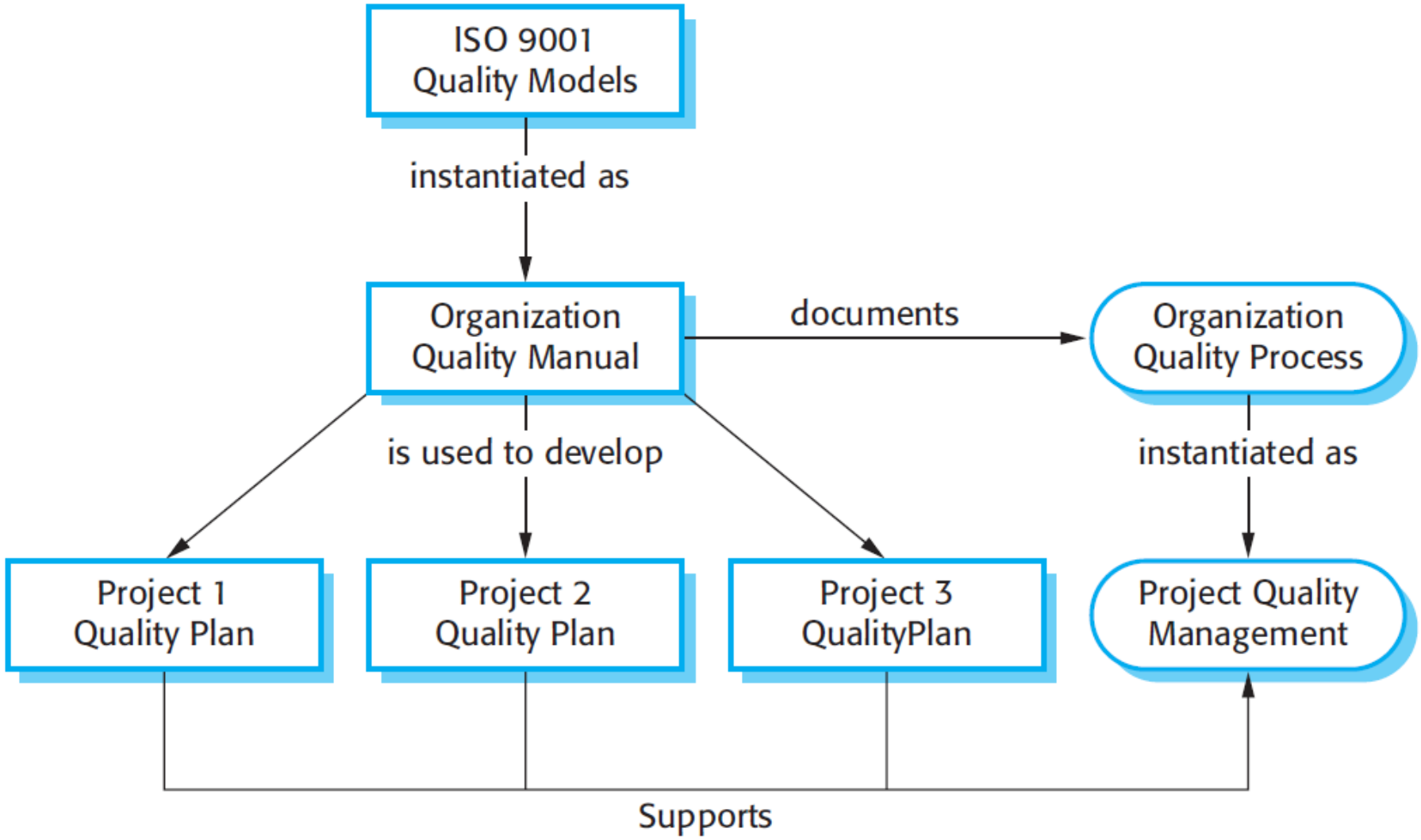
- An international set of standards for quality management
- Applicable to a range of organizations from manufacturing to service industries

- 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.



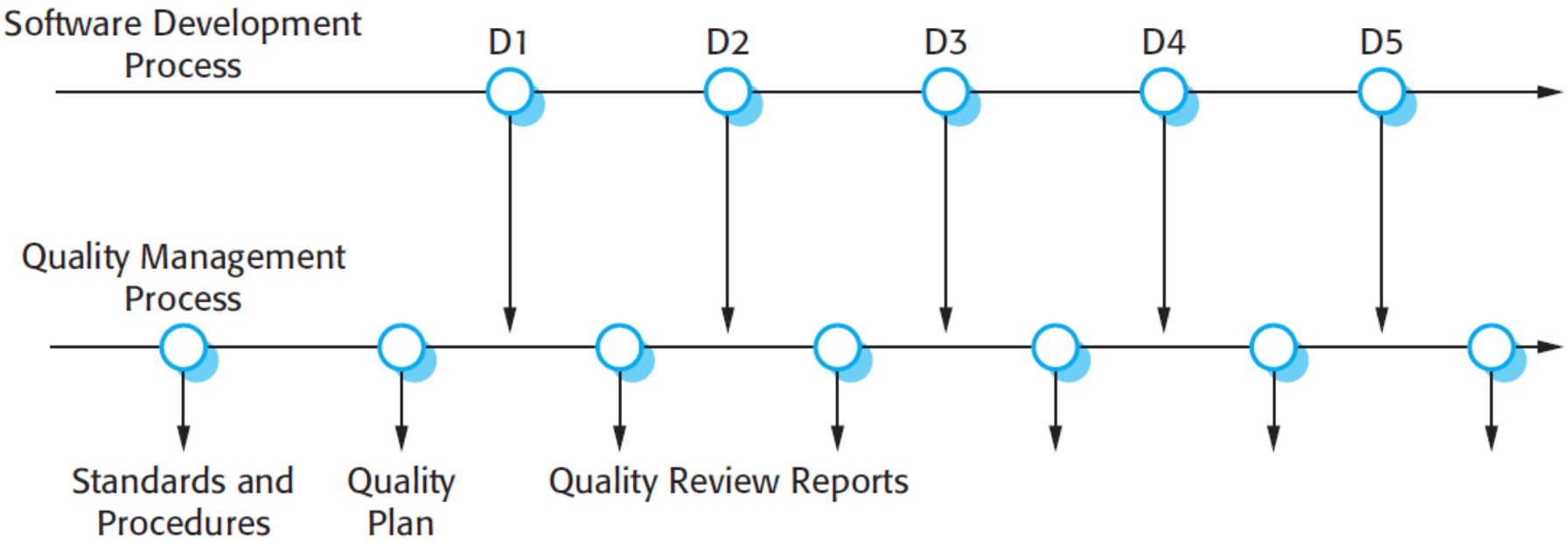
["Software Engineering." Ian Sommerville. (2010)]

ISO 9001 and quality management



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

Quality management and software development



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

1. The context of software engineering

1.1. Introduction and overview

1.2. Characteristics of software systems

1.3. Factors affecting the design of a software system

1.4. Introduction to embedded systems (Guest lecture)

Next lecture!