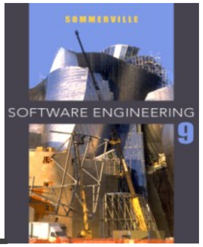
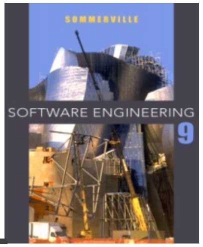


Course Organization

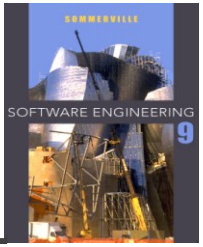
Lecture 1/Part 1

Outline



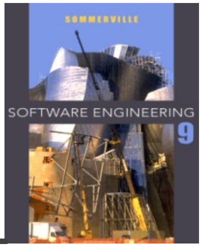
- ✧ About me
- ✧ About the course
  - Lectures
  - Seminars
  - Evaluation
- ✧ Literature

About me:
Ing. RNDr. Barbora Bühnová, Ph.D.



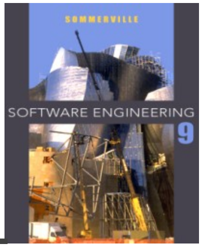
- Industrial experience
- Research
  - Quality of software architecture
  - LaSArIS
- Teaching
  - Courses on UML, Java, .NET, Automata and grammars, Algorithm design, and others
- Colaboration with students
  - Seminar tutoring
  - Bachelor/Master theses

About the course:
PB007 Software Engineering I



- Lectures
    1. Software process, role of the UML language.
    2. **Functional requirements** specification, UML Use Case diagram.
    3. **Nonfunctional requirements** specification, UML Activity diagram.
    4. System analysis and design, structured vs. object-oriented A&D.
    5. **Object oriented analysis**, UML Class, Object and Interaction diagrams.
    6. **Structured analysis**, data modelling, ERD.
    7. **System design** and attributes of a high-quality design, UML State diagram.
    8. **Software architecture**, UML Component and Deployment diagram.
    9. **User interface design**.
    10. **Testing**, verification and validation.
    11. **Operation**, maintenance and system evolution.
    12. Software development management - processes, tools and frameworks.
    13. Advanced software engineering techniques.

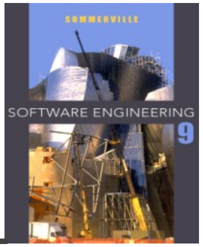
About the course:
PB007 Software Engineering I



✧ Seminars

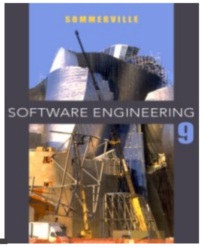
1. Visual Paradigm introduction, project assignment.
2. Project start, initial **Use Case diagram**.
3. Detailed **Use Case diagram**, textual specification of UC
4. Specification of use cases (textual if not finished, **Activity diagram**).
5. Analytical **Class diagram**, **Object diagram**.
6. Finalization of analytical **Class diagram**, **Use Case diagram** update.
7. Data modelling, **Entity Relationship diagram**.
8. Refinement of use cases with **Interaction diagrams**.
9. Finalization of **Interaction diagrams**, **Class diagram** update.
10. **State diagram**.
11. Design-level **Class diagram**, interfaces, implementation details.
12. **User interface design**.
13. Packages, **Component diagram**, **Deployment diagram**.

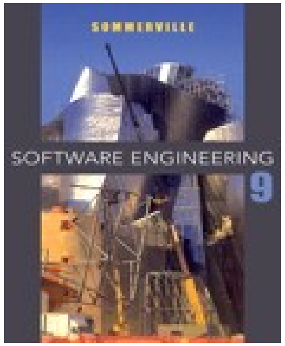
About the course:
PB007 Software Engineering I



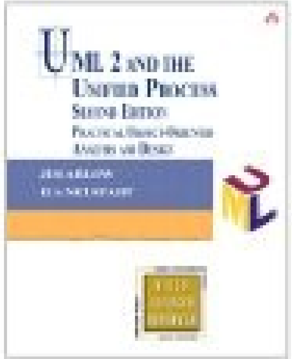
- Lectures
  - 13 teaching weeks + 1 week free
- Seminars
  - Team project on UML modelling
  - 2-3 students per team
  - Obligatory attendance and weekly task delivery
  - Penalty for absence (-5/-10) and task delivery (-5/-10 points)
- Evaluation
  - Exam = test (50 points) + on-site modelling (50 points)
  - Grades: 90-100 A, 80-89 B, 70-79 C, 60-69 D, 50-59 E, 0-49 F

Literature

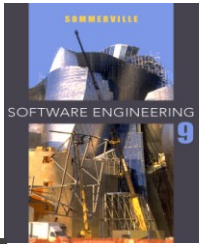




- ✧ Software Engineering, 9/E
  - Author: Ian Sommerville
  - Publisher: Addison-Wesley
  - Copyright: 2011



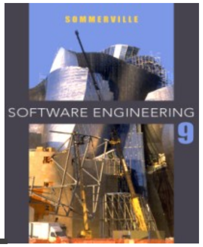
- ✧ UML 2 and the Unified Process, 2/E
  - Author: Jim Arlow and Ila Neustadt
  - Publisher: Addison-Wesley
  - Copyright: 2005



Software process

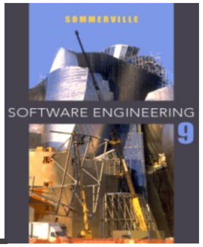
Lecture 1/Part 2

Outline



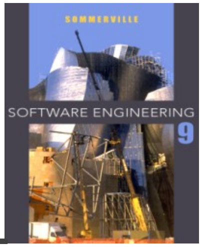
- Software engineering
- Software process activities
- Software process models

Software engineering



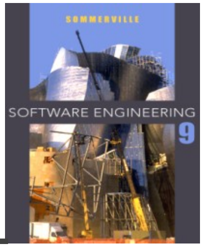
- The **economies** and **human lifes** of ALL developed nations are dependent on software.
- More and more systems are software controlled
- Software engineering is concerned with **theories**, **methods** and **tools** for professional software development.
- Software engineering is concerned with **cost-effective** development of **high-quality** software systems .

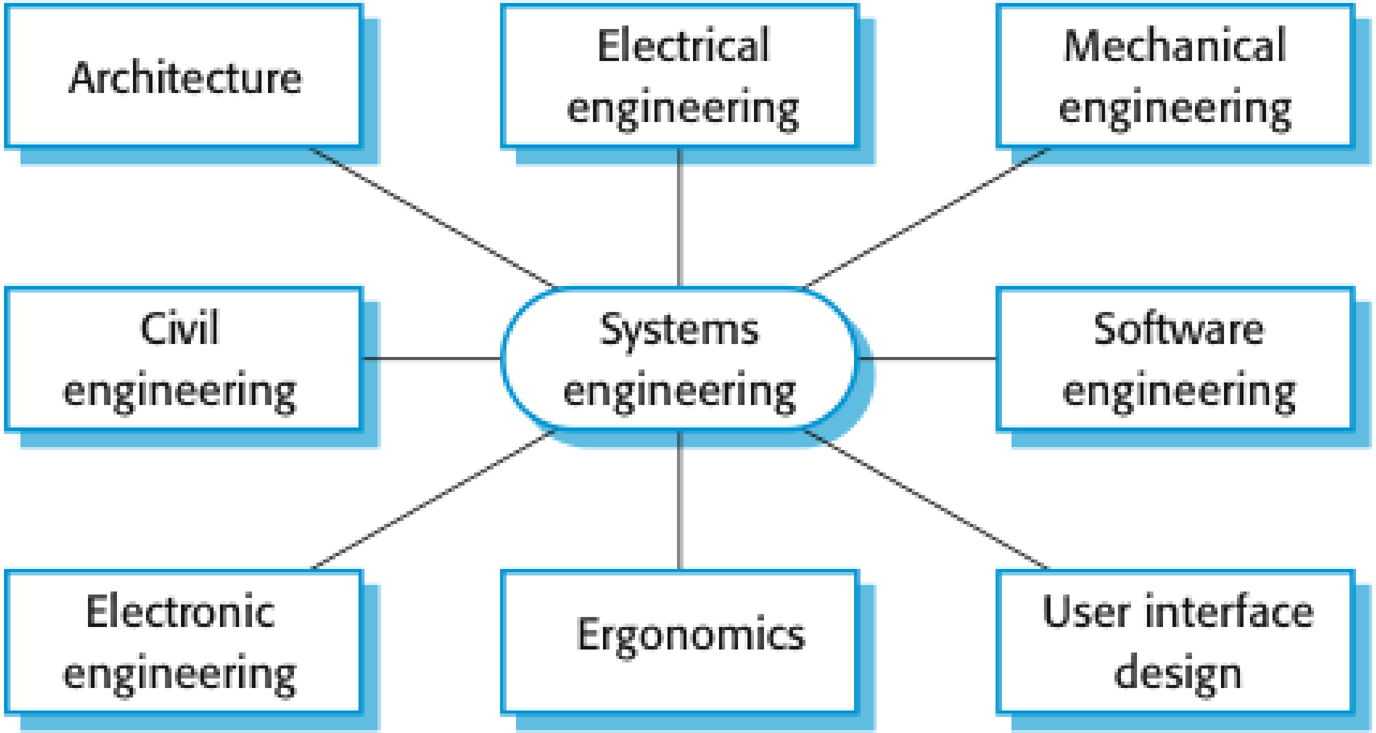
Frequently asked questions about software engineering



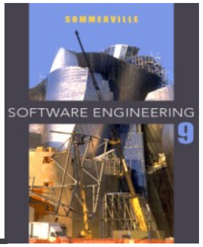
Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable (among others).
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

Software versus System engineering



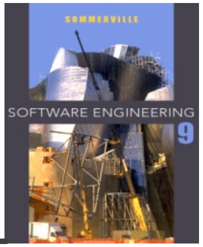


Software products



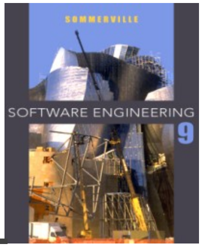
- ✧ Generic products
  - Stand-alone systems that are marketed and sold to **any customer** who wishes to buy them.
  - **Examples** – PC software such as graphics programs, project management tools; CAD software.
- ✧ Customized products
  - Software that is commissioned by a **specific customer** to meet their own needs.
  - **Examples** – embedded control systems, air traffic control software, traffic monitoring systems.

Application types



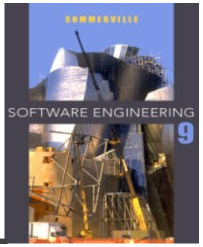
- Stand-alone applications
- Interactive transaction-based applications
- Embedded control systems
- Batch processing systems
- Entertainment systems
- Systems for modeling and simulation
- Data collection and monitoring systems
- Systems of systems

Software engineering fundamentals



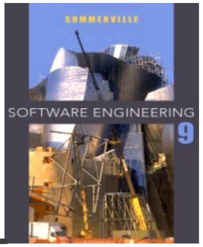
- Some **fundamental principles** apply to all types of software system, irrespective of the development techniques used:
  - Systems should be developed using a **managed and understood development process**. Of course, different processes are used for different types of software.
  - **Dependability and performance** are important for all types of system.
  - Understanding and managing the **software specification and requirements** (what the software should do) are important.
  - Where appropriate, you should **reuse software** that has already been developed rather than write new software.

The software process



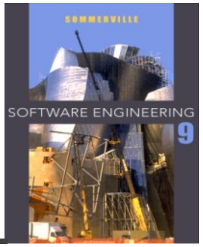
- ✧ A structured set of activities required to develop a software system.
- ✧ Many different software processes but all involve:
  - **Specification**
  - Development
    - **Analysis and design**
    - **Implementation**
  - **Validation**
  - **Evolution**
- ✧ A software process model is an abstract representation of a process – from some particular **perspective**.

Software process activities



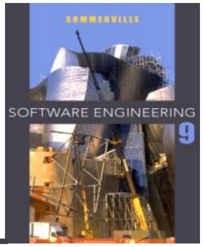
- **Software specification**, where customers and engineers define the software and the constraints on its operation.
- **Software analysis and design**, where the requirements are refined into system design.
- **Software implementation**, where the software is implemented.
- **Software validation**, where the software is checked to ensure that it is what the customer requires.
- **Software evolution**, where the software is modified to reflect changing customer and market requirements.

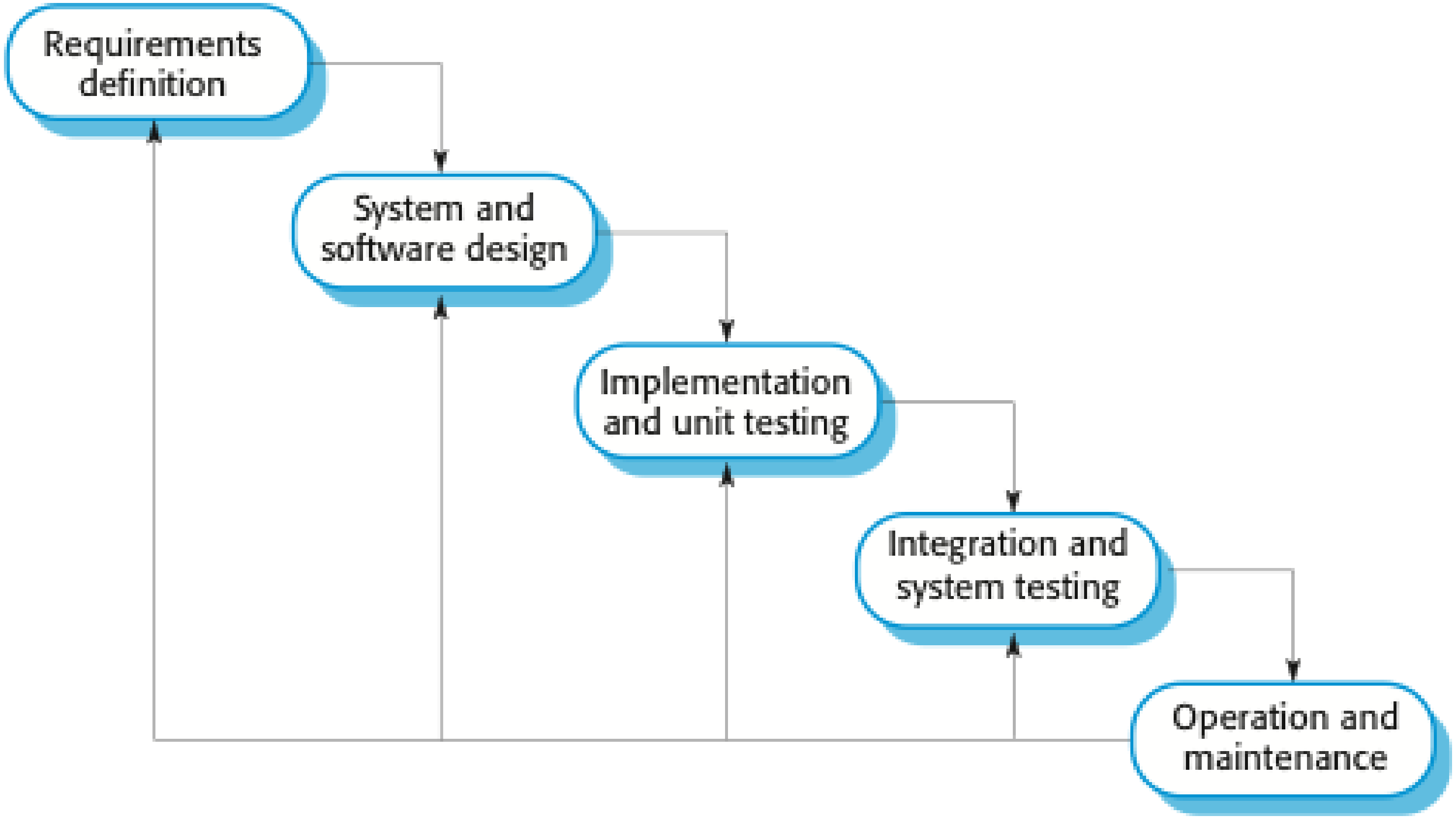
Software process models



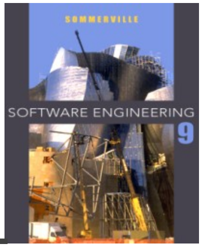
- **The waterfall model**
  - Plan-driven model. Separate and distinct phases of specification and development.
- **Incremental development**
  - Specification, development and validation are interleaved. May be plan-driven or agile (respecting agile development principles).
- **Reuse-oriented software engineering**
  - The system is assembled from existing components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements **from all of these models**.

The waterfall model



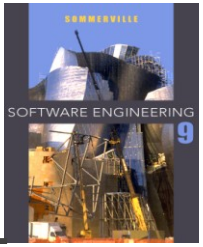

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

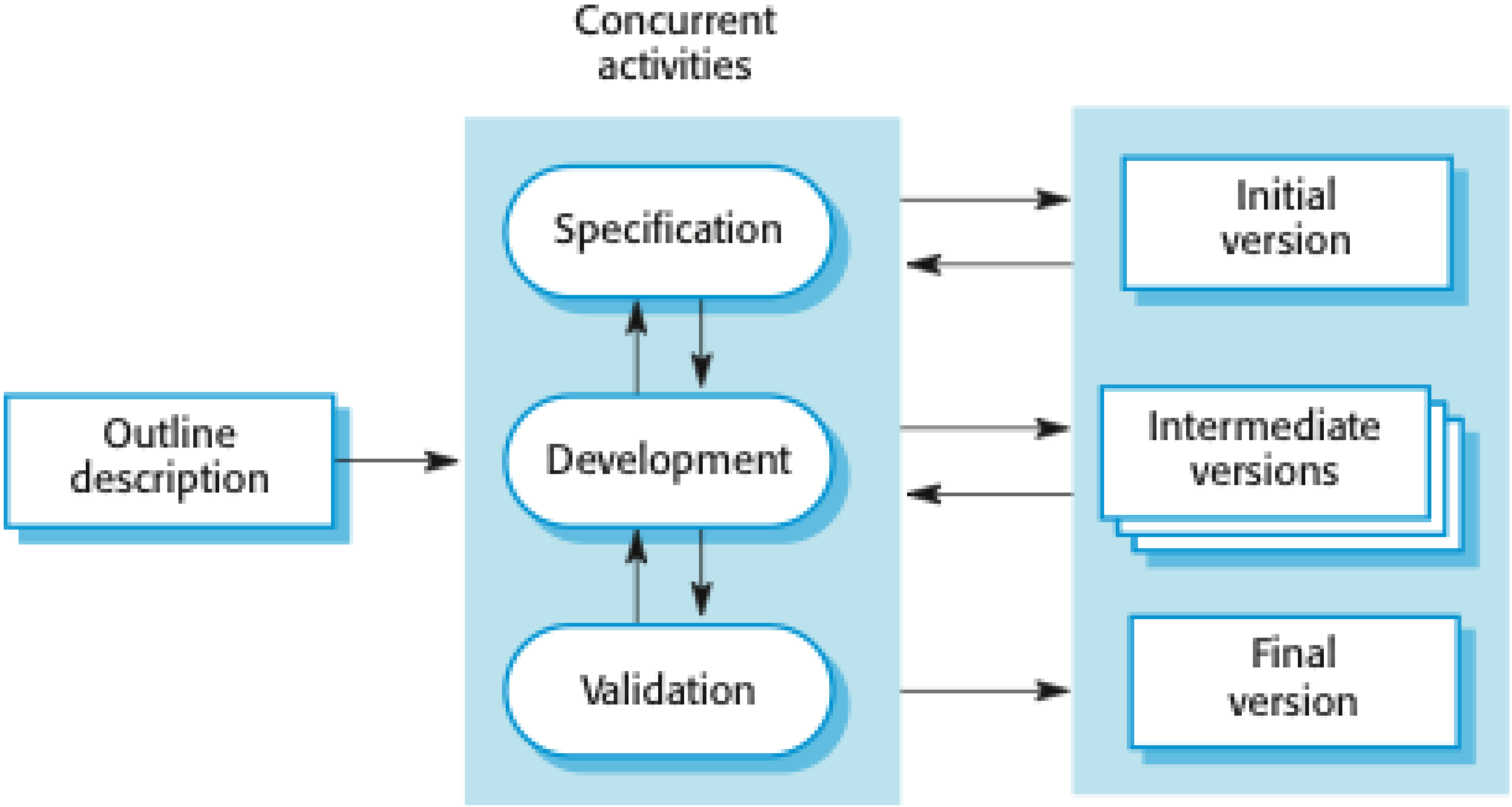
Waterfall model problems



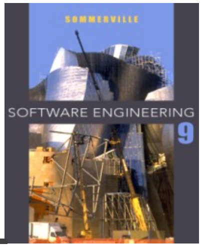
- **Inflexible** partitioning of the project into distinct stages makes it difficult to respond to **changing customer requirements**.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for **large systems engineering projects** where a system is developed at several sites, and for **generic products**.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Incremental development



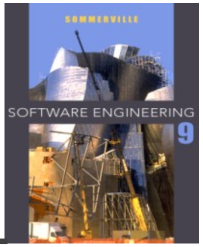


Incremental development benefits



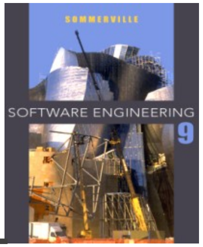
- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development problems



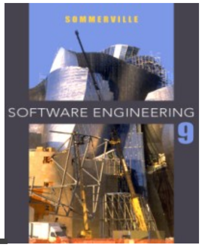
- The process is not visible.
  - Managers need **regular deliverables** to measure progress. If systems are developed quickly, it is not cost-effective to produce **documents** that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on **refactoring** to improve the software, regular change tends to **corrupt its structure**. Incorporating further software changes becomes increasingly **difficult and costly**.

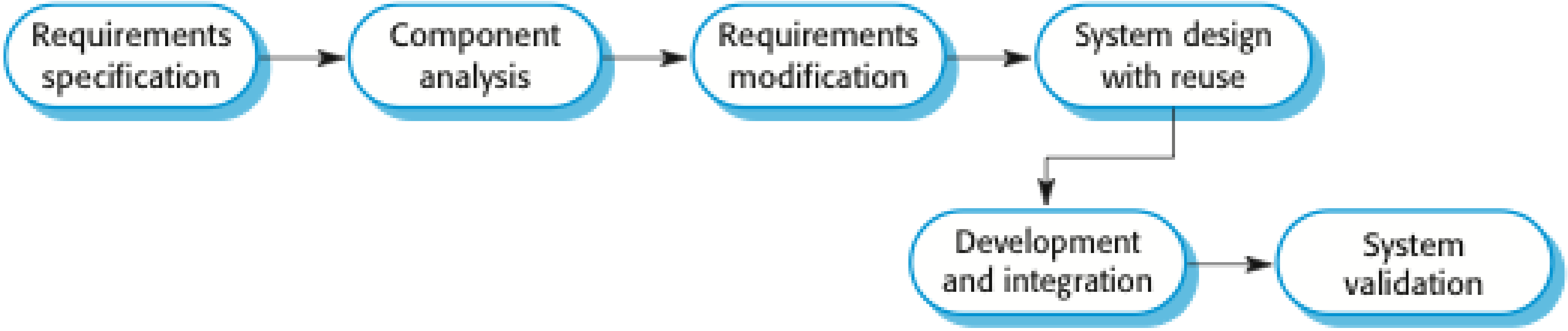
Reuse-oriented software engineering



- Based on **systematic reuse** where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.
- Reuse is now the standard approach for building many types of business system

Reuse-oriented software engineering





Requirements specification

Component analysis

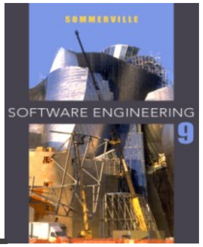
Requirements modification

System design with reuse

Development and integration

System validation

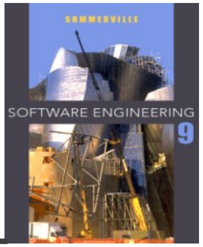
Key points



- There are many different types of system and each requires appropriate software engineering tools and techniques for their development.
  - The fundamental ideas of software engineering are applicable to all types of software system.

- Software engineering is an engineering discipline that is concerned with all aspects of software production.
  - The high-level activities of specification, development (analysis and design, and implementation), validation and evolution are part of all software processes.

Key points

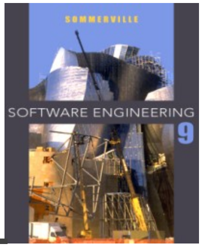


- Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes. Examples of these general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.

UML in Software Development

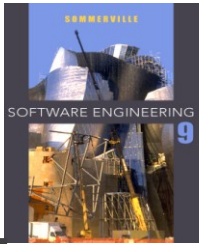
Lecture 1/Part 3

Outline



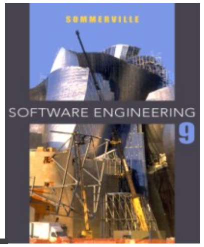
- System modeling
- Structural models
- Interaction models
- Behavioral models

System modeling



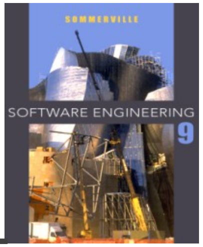
- System modeling is the process of developing **abstract models of a system**, with each model presenting a different view or **perspective** of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the **Unified Modeling Language (UML).**
- System modelling helps the analyst to **understand the functionality** of the system and models are used to **communicate with colleagues and customers**.

System perspectives



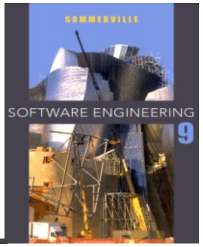
- An **external perspective**, where you model system boundary, the context and/or environment of the system.
- A **structural perspective**, where you model the organization of a system or the structure of the data that is processed by the system.
- An **interaction perspective**, where you model the interactions between a system and its environment, or between the components of a system.
- A **behavioral perspective**, where you model the dynamic behavior of the system and how it responds to events.

UML diagram types



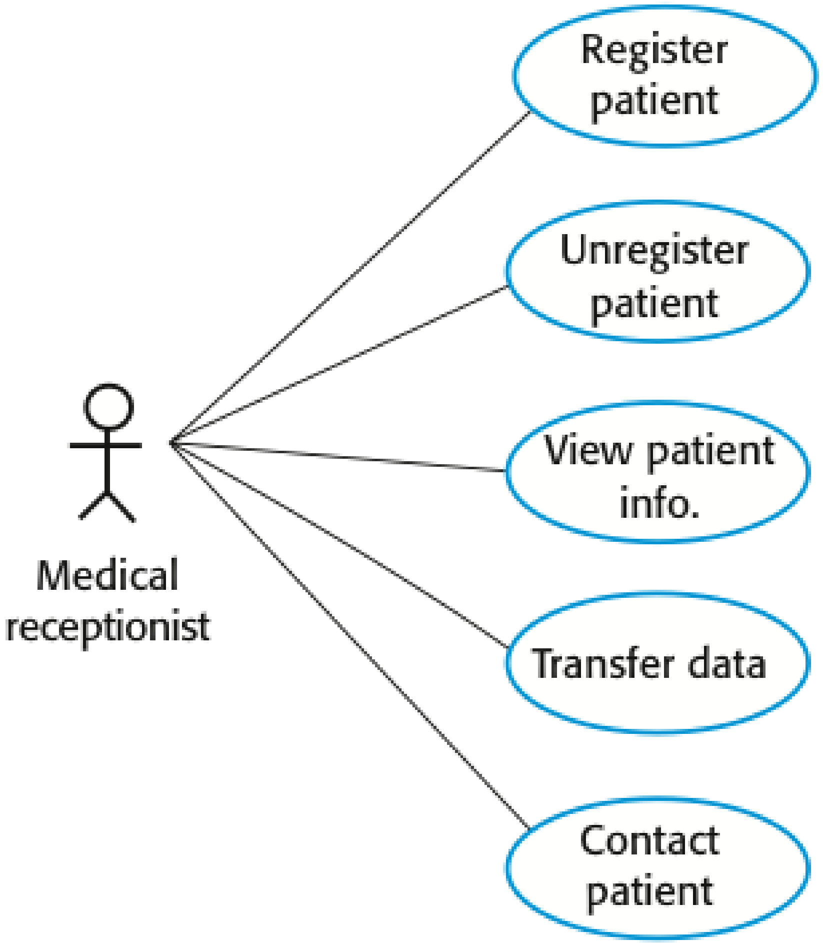
- External perspective
  - **Use case diagram**
- Structural perspective
  - **Class diagram**, Object diagram, Component diagram, Package diagram, Deployment diagram, Composite structure diagram
- Interaction perspective
  - **Sequence diagram**, Communication diagram, Interaction overview diagram, Timing diagram
- Behavioral perspective
  - **Activity diagram**, State diagram

Popular UML diagrams

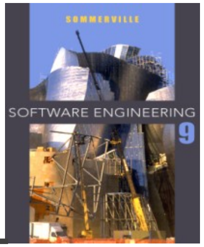


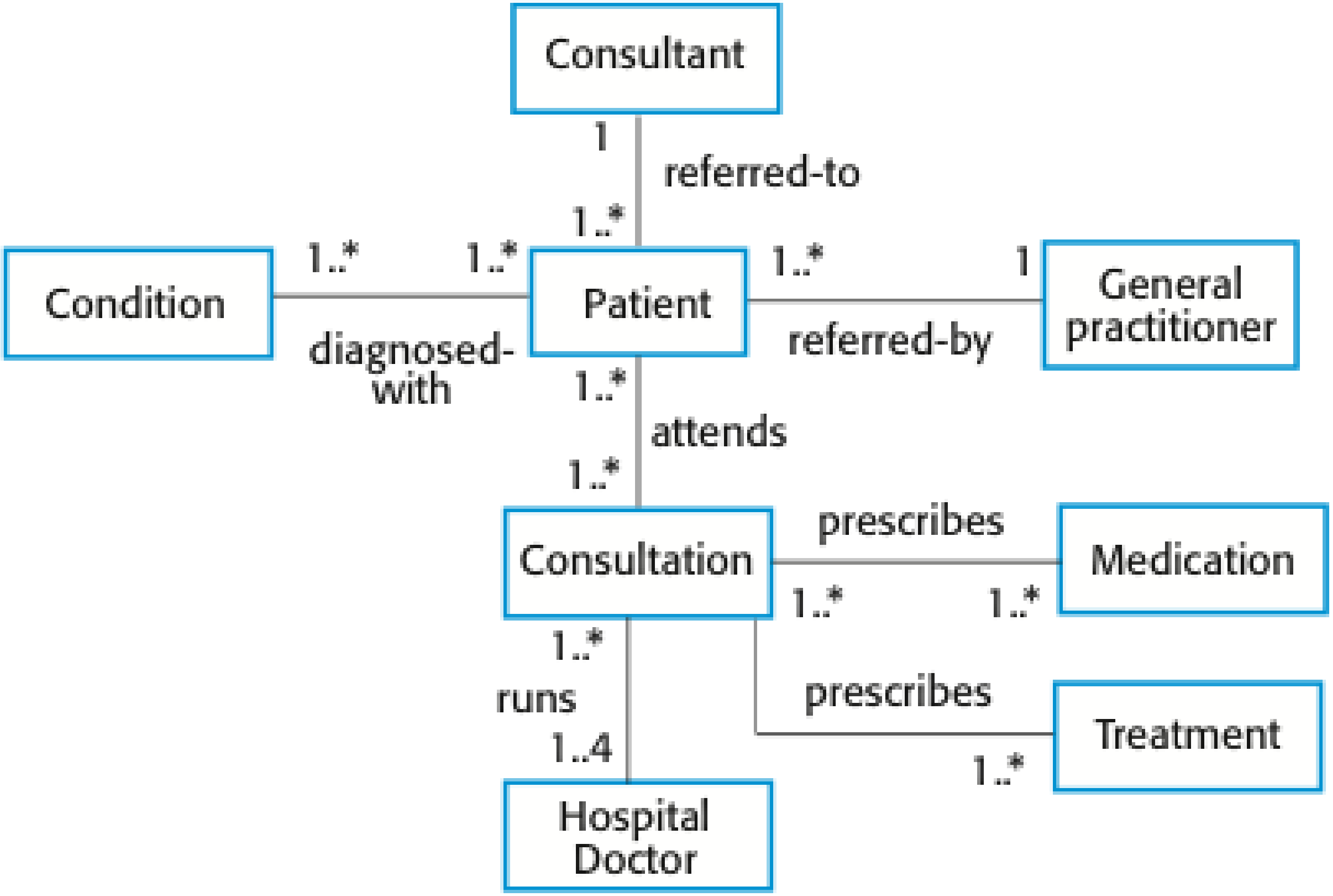
- **Use case diagrams**, which show the interactions between a system and its environment.
- **Class diagrams**, which show the object classes in the system and the associations between these classes.
- **Sequence diagrams**, which show interactions between actors and the system and between system components.
- **Activity diagrams**, which show the activities involved in a process or in data processing.

UML Use case diagram: Medical receptionist in health care system

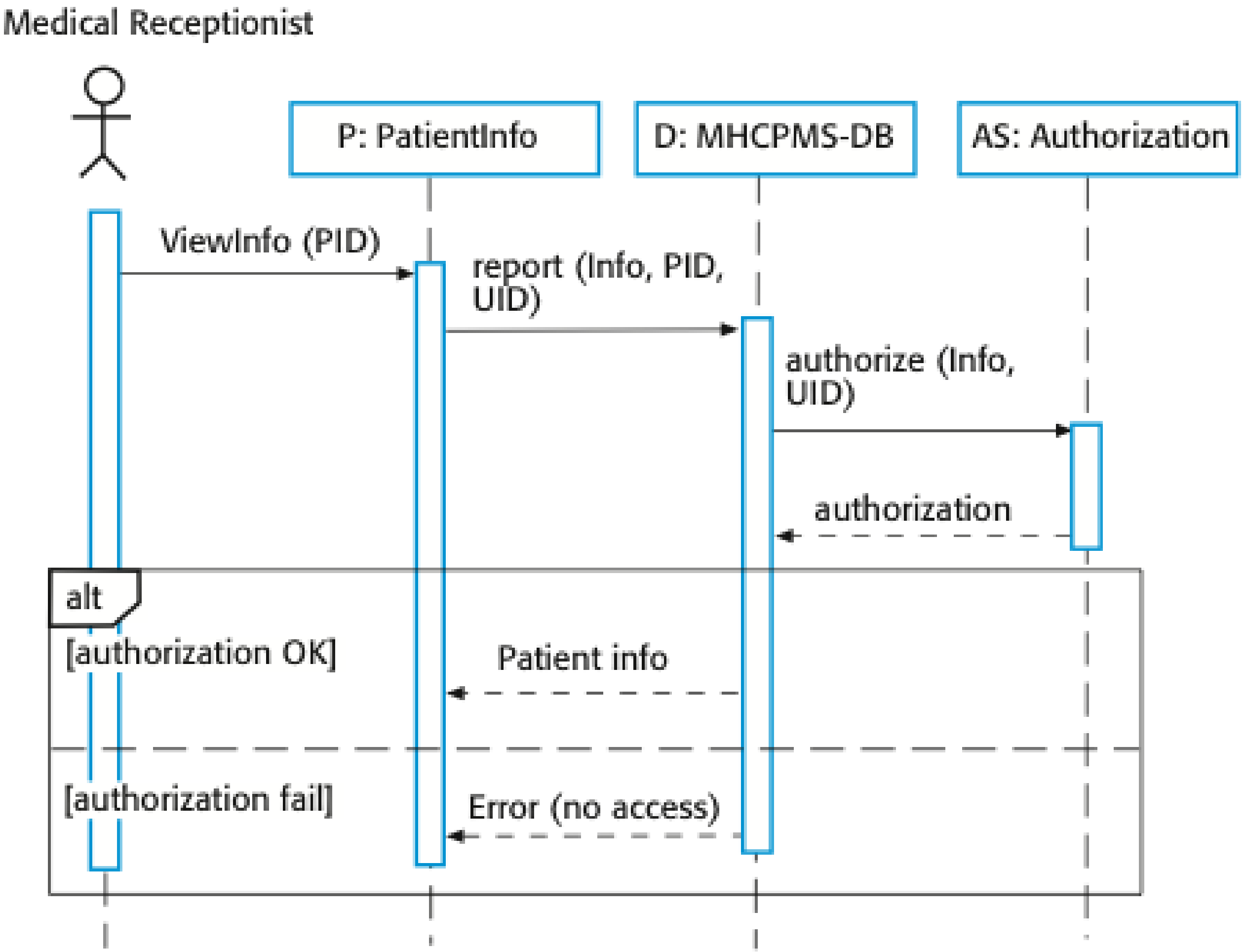


UML Class diagram: Health care system

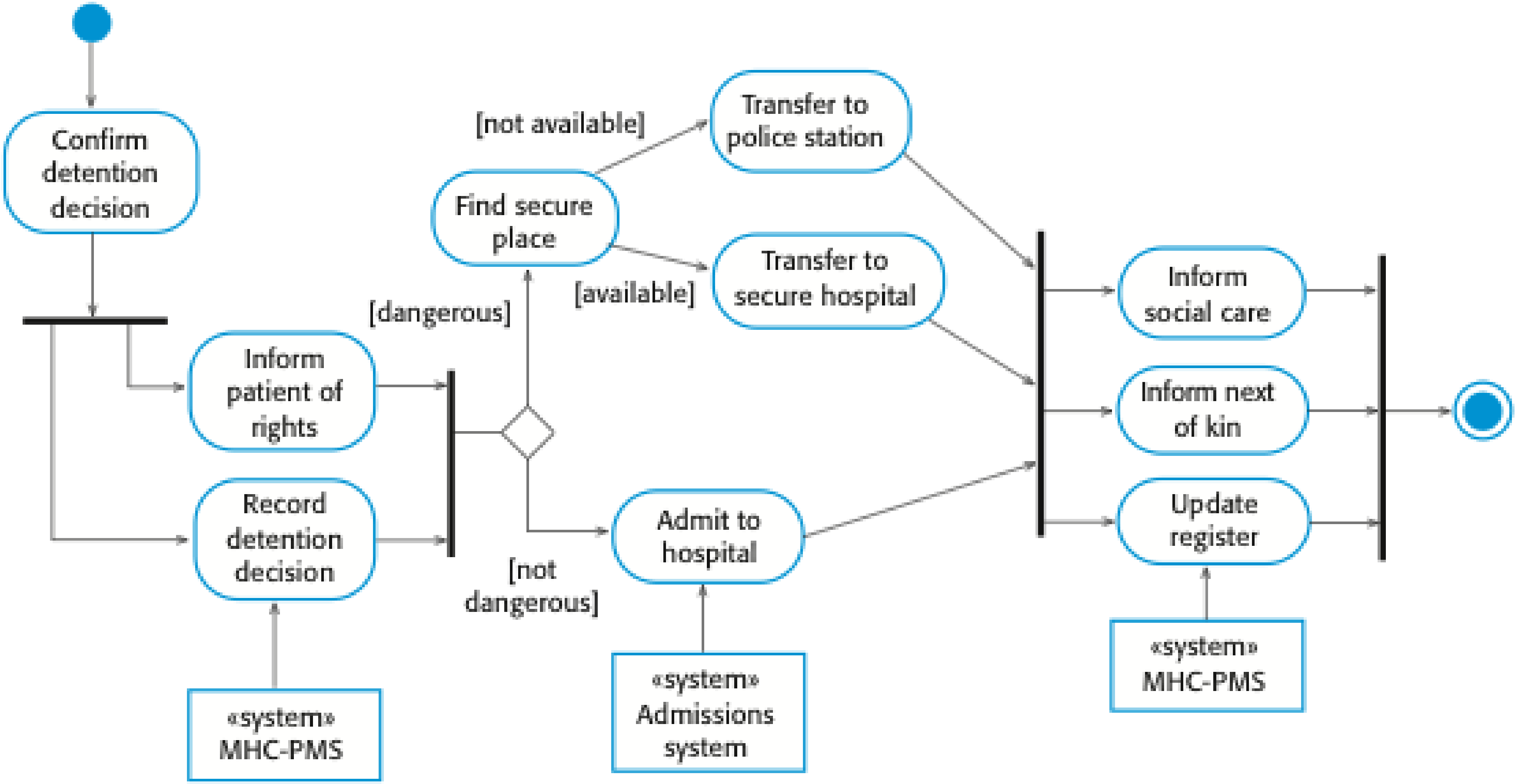




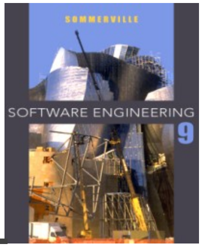
UML Sequence diagram: View patient information in health care system



UML Activity diagram: Process model of involuntary detention



Key points



- A model is an **abstract view** of a system that ignores system details. Complementary system models can be developed to show the system's **context**, **structure**, **behavior** and **interactions**.
- **Context models** show how a system that is being modeled is positioned in an environment with other systems.
- **Structural models** show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.
- **Interaction models** are used to describe the interactions between system elements and **Behavioral models** to detail the internal dynamic behavior of system elements/processes.