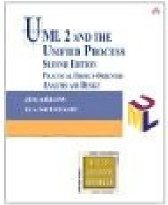

UML 2 and the Unified Process
Second Edition

SOMMERVILLE
SOFTWARE ENGINEERING
9

Course Organization

Lecture 1/Part 1

Outline

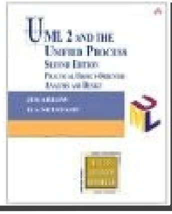
✧ About the lecturer

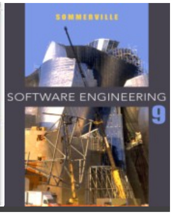
✧ About the course

- Lectures
- Seminars
- Evaluation

✧ Literature

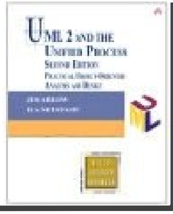
About the lecturer:
Ing. RNDr. Barbora Bühnová, Ph.D.





- Industrial experience
- Research
  - Quality of software architecture
  - Lab of Software Architecture and Information Systems (LaSArIS)
- Teaching
  - Courses on UML, architecture design, programming, algorithm design, automata and grammars, and others
- Collaboration with students
  - Seminar tutoring
  - Bachelor/Master theses

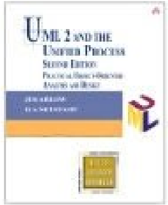
About the course: PB007 Software Engineering I



✧ 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 State diagram.
6. **Structured analysis**, data modelling, ERD.
7. **High-level design**, UML Class diagram in design.
8. **Low-level design and implementation** issues, UML Interaction diagrams
9. **Architecture design**, UML Package, Component and Deployment diagram.
10. **Testing**, verification and validation.
11. **Operation**, maintenance and system evolution.
12. Software development management.
13. Advanced software engineering techniques.

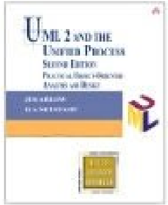
About the course:
PB007 Software Engineering I



✧ Seminars

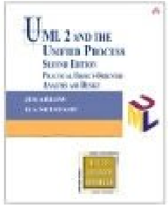
1. Visual Paradigm introduction, project assignment.
2. Project start, initial **Use Case diagram**.
3. Detailed **Use Case diagram**, textual specification of UC
4. Specification of use cases (textual if not finished, **Activity diagram**).
5. Analytical **Class diagram**, **Object diagram**.
6. Finalization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

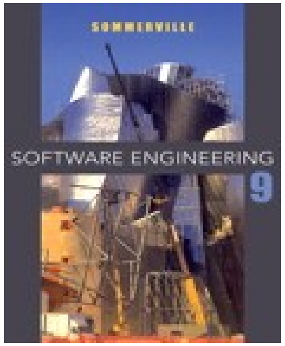
About the course:
PB007 Software Engineering I



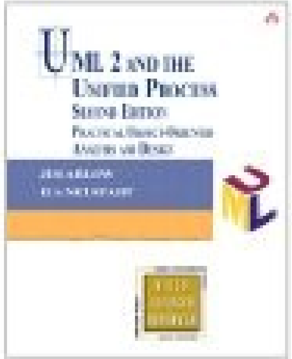
- **Prerequisites**
  - Basic knowledge of object oriented programming
- **Lectures**
  - 13 teaching weeks + 1 week free (28.10.2014)
- **Seminars**
  - 12 teaching weeks + 1 week final project discussion
  - Team project on UML modeling, teams of 3 students (or less)
  - Obligatory attendance (one absence ok) and weekly task delivery
  - Simple test at the beginning of each seminar (starting in Week 03)
  - Penalty for extra absence (-5 points) and late task delivery (-5 p.)
- **Evaluation**
  - Seminar = project YES/NO, test (20 points) and penalty recorded in IS notebook
  - Exam = test (35 points) + on-site modelling (35 points)
  - Grades: F<50, 50<=E<58, 58<=D<66, 66<=C<74, 74<=B<82, 82<=A

Literature

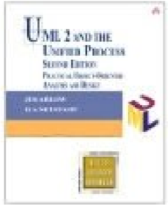




- ✧ Software Engineering, 9/E
  - Author: Ian Sommerville
  - Publisher: Addison-Wesley
  - Copyright: 2011



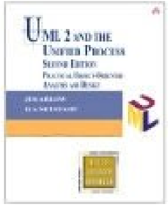
- ✧ UML 2 and the Unified Process, 2/E
  - Author: Jim Arlow and Ila Neustadt
  - Publisher: Addison-Wesley
  - Copyright: 2005



Software process

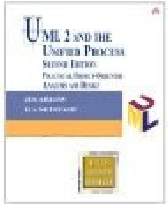
Lecture 1/Part 2

Outline



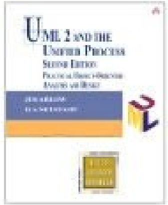
- Software engineering
- Software process activities
- Software process models

Software and system engineering



- The **economies** and **human lifes** of ALL developed nations are dependent on software.
- 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.
- **System engineering** is concerned with all aspects of computer-based systems development including hardware, software and process engineering.

Software products



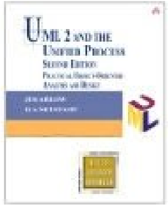
✧ Generic products

- 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

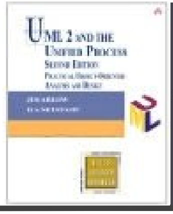
- Software that is commissioned by a **specific customer** to meet their own needs.
- **Examples** – embedded control systems, air traffic control software, traffic monitoring systems.

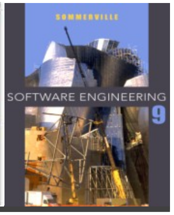
Application types



- ✧ Stand-alone desktop applications
- ✧ Interactive web-based applications
- ✧ Embedded control systems
- ✧ Batch processing systems
- ✧ Computer games
- ✧ Systems for modeling and simulation
- ✧ Data collection and monitoring systems
- ✧ Systems of systems

Software engineering fundamentals

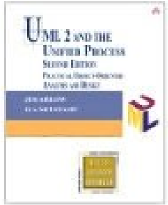


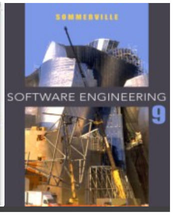


✧ Some **fundamental principles** apply to all types of software system, irrespective of the development techniques used:

- 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:

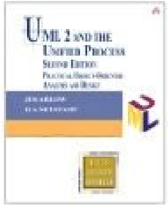
- **Specification**
- **Analysis and design**
- **Implementation**

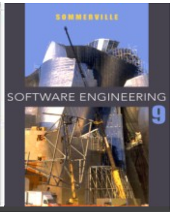
} **Development**

- **Validation and verification**
- **Evolution**

✧ Is the analysis and design always involved?

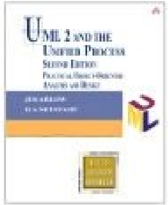
Software process activities





- **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 and verification**, 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**

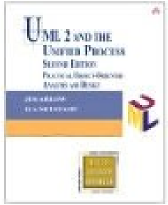
- Specification, development and validation are interleaved. May be plan-driven or agile.

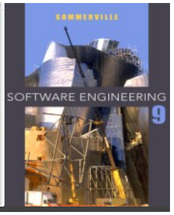
✧ **Reuse-oriented software engineering**

- 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 many different models**.

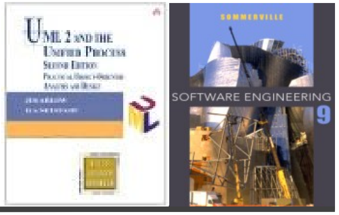
Plan-driven and agile development

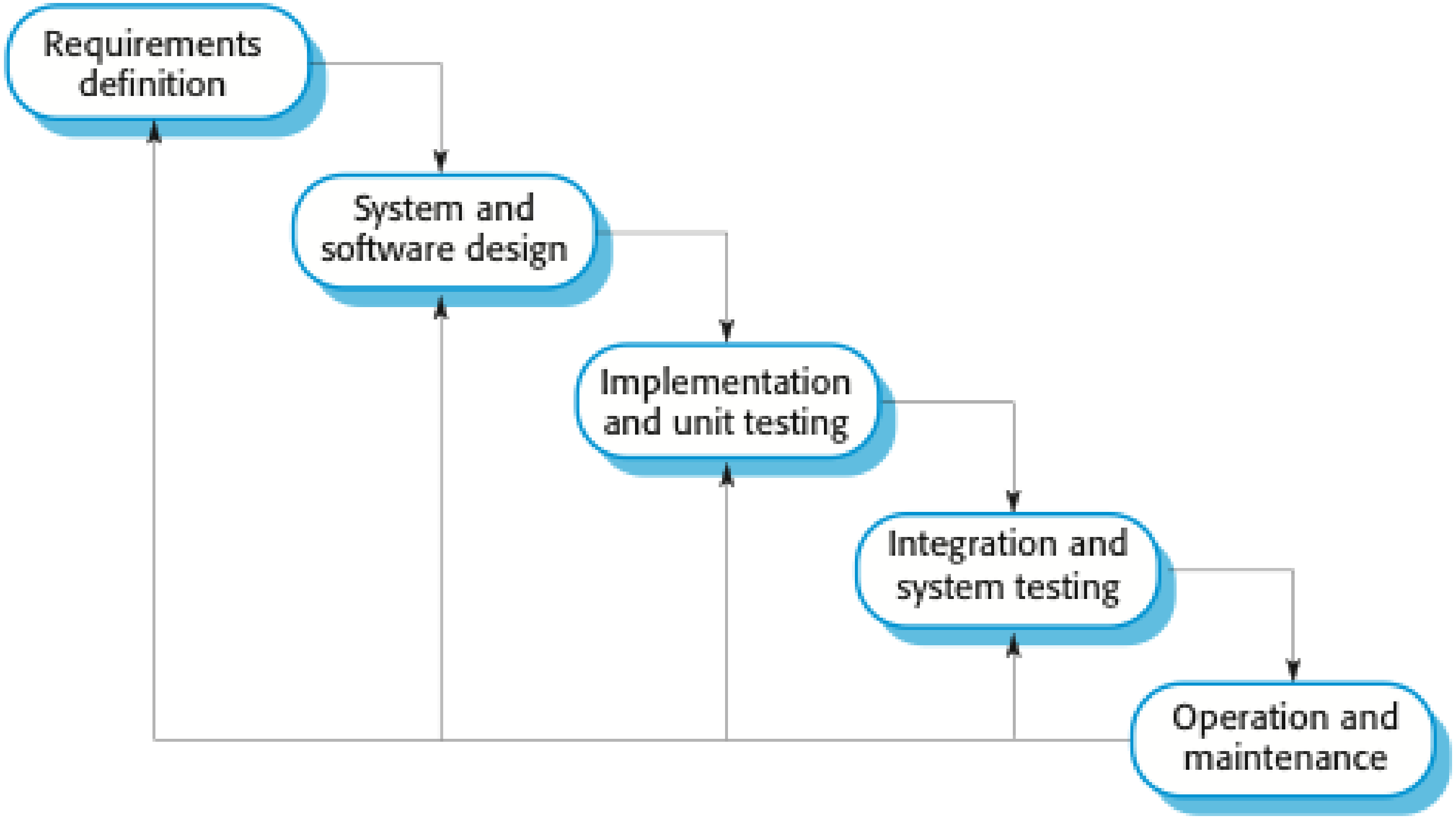




- ✧ Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the **outputs** to be produced at each of these stages **planned in advance**.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
- ✧ Agile development
  - Specification, design, implementation and testing are inter-leaved and the **outputs** from the development process are **decided through a process of negotiation** during the software development process.

The waterfall model





Requirements definition

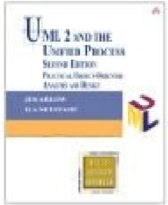
System and software design

Implementation and unit testing

Integration and system testing

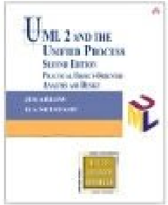
Operation and maintenance

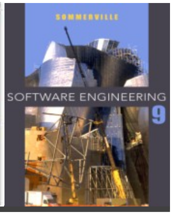
Waterfall model benefits and problems



- ✧ The waterfall model is mostly used for **large system engineering projects** where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.
- ✧ Suitable for new versions of **generic products.**
  - Well understood context, stable requirements.
- ✧ The process makes it difficult to respond to **changing customer requirements**.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes can be limited.

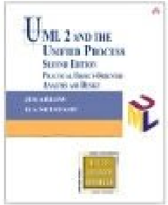
Software prototyping





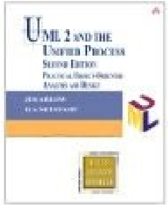
- A prototype is an initial version of a system used to demonstrate concepts and try out design options.
- A prototype can be used in:
  - The requirements engineering process to help with **requirements elicitation**, consistency checking and validation;
  - In design processes to **explore design options** and develop a **UI design**;
- Prototypes often have **poor internal structure** and thus should not become the foundation of the final system.

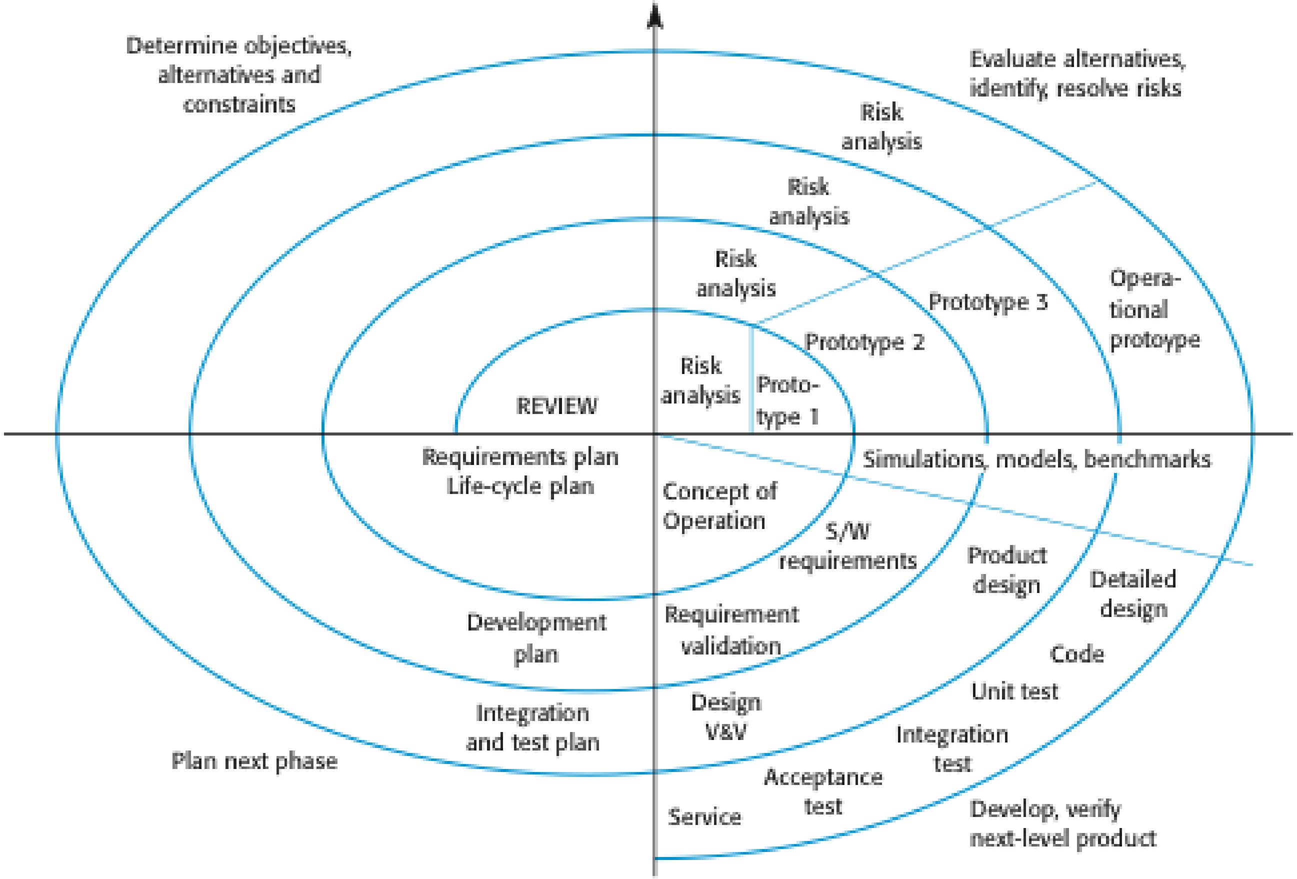
Boehm's spiral model



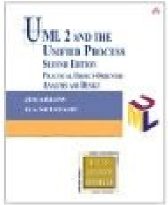
- **Process is represented as a spiral** rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- **No fixed phases** such as specification or design - loops in the spiral are chosen depending on what is required.
- **Risks** are explicitly assessed and resolved throughout the process.

Boehm's spiral model of the software process



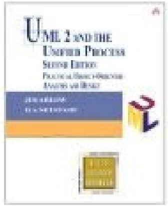


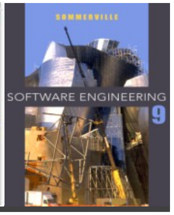
Spiral model sectors



- Objective setting
  - Specific objectives for the phase are identified.
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks.
- Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- Planning
  - The project is reviewed and the next phase of the spiral is planned.

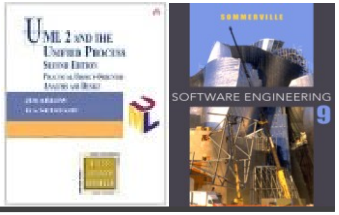
The Rational Unified Process



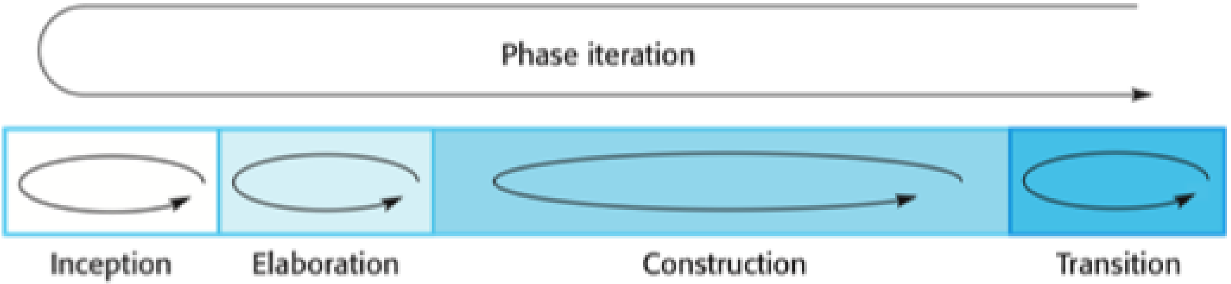


- A modern generic process commonly associated with the Unified Modeling Language (UML).
- Brings together aspects of a number of generic process models discussed in this lecture. Which ones?
- Normally described from 3 perspectives
  - A **dynamic perspective** that shows phases over time;
  - A **static perspective** that shows process activities;
  - A **practice perspective** that suggests good practices to be used during the process.

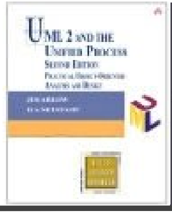
Phases in the Rational Unified Process

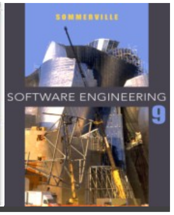


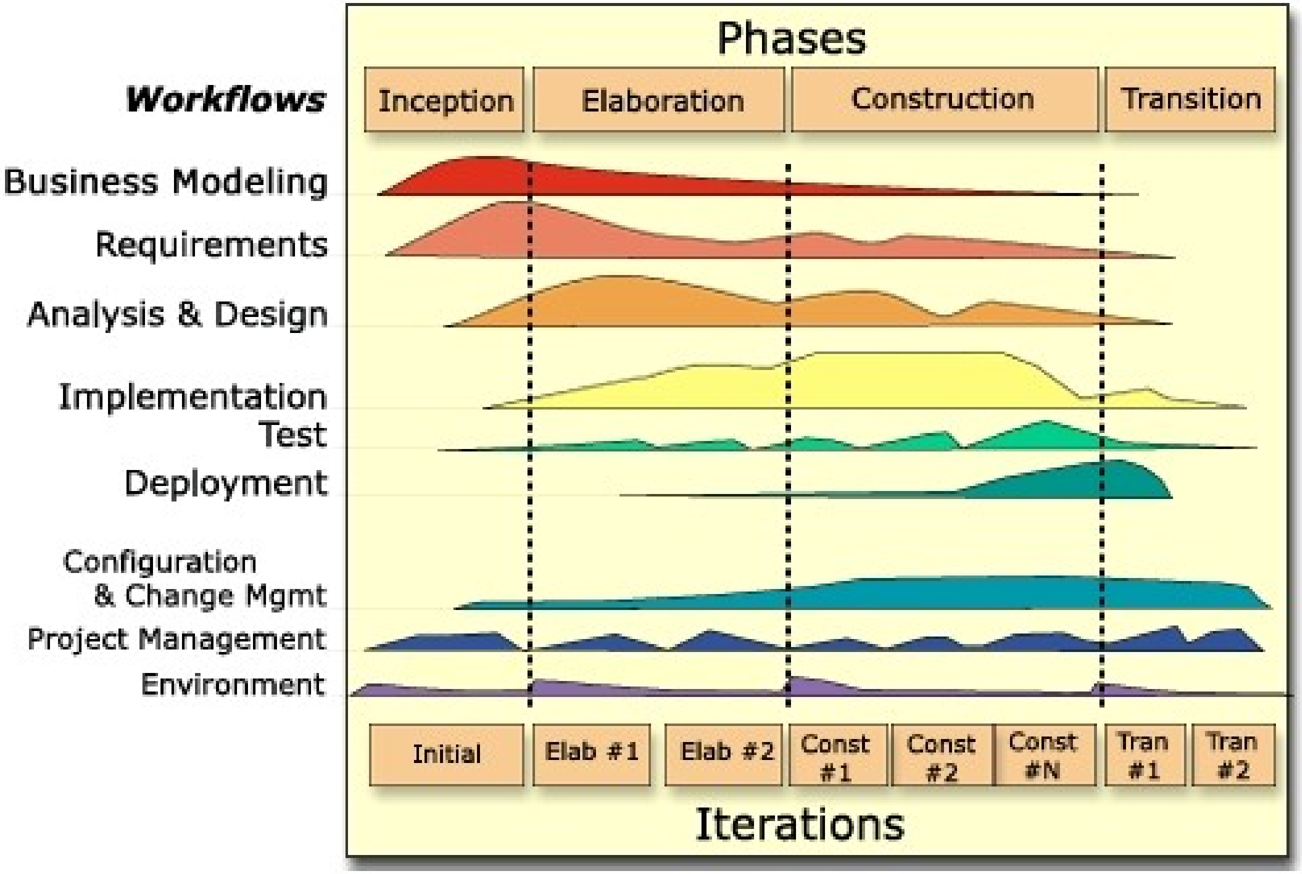
- ✧ Inception
  - Establish the business case for the system.
- ✧ Elaboration
  - Develop understanding of the problem domain and system architecture.
- ✧ Construction
  - System design, programming and testing.
- ✧ Transition
  - Deploy the system in its operating environment.



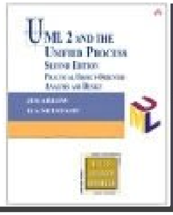
RUP process architecture

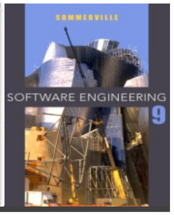




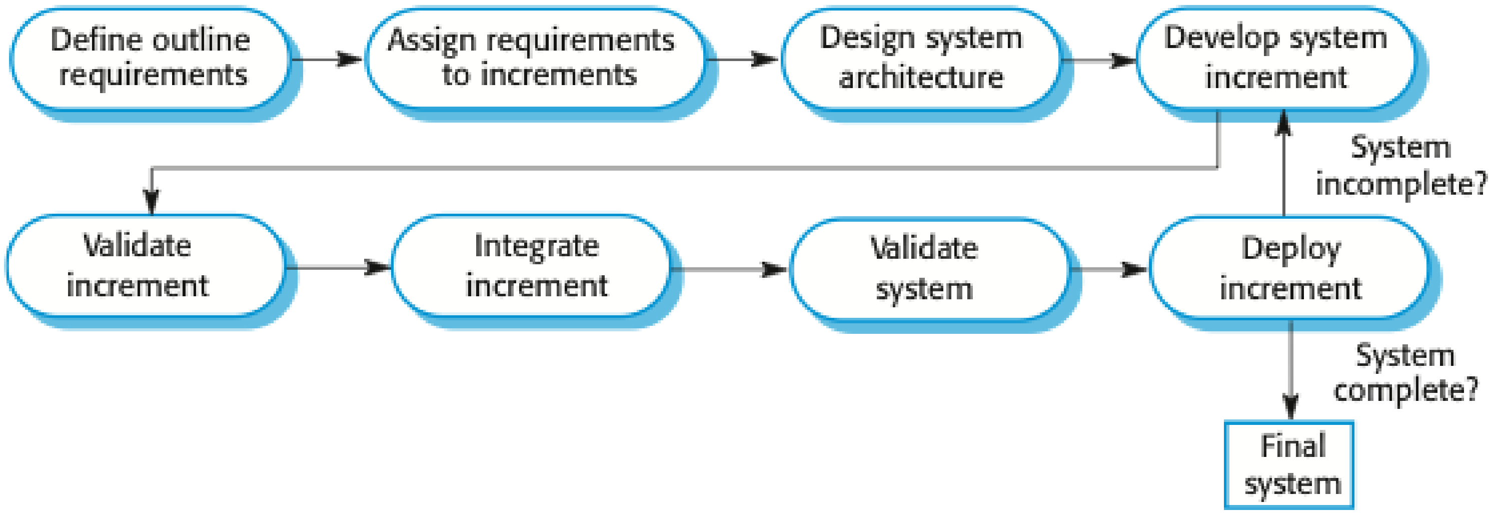


Iterative and incremental development

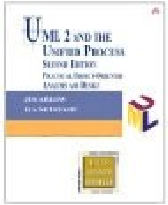


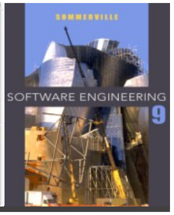


✧ What is the difference between the two?



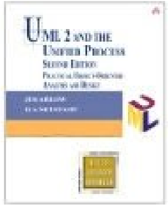
Incremental delivery





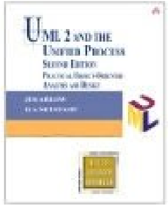
- ✧ Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with **each increment delivering part of the required functionality**.
- ✧ User requirements are **prioritised** and the highest priority requirements are included in early increments.
- ✧ Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

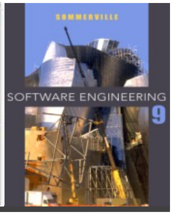
Incremental development benefits



- ✧ **Customer value** can be delivered with each increment so system functionality is available earlier.
- ✧ Early increments act as a **prototype** to help elicit requirements for later increments.
- ✧ **Lower risk** of overall project failure.
- ✧ The **highest priority** system services tend to receive the most attention (design, testing, etc.).

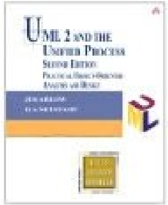
Incremental development problems

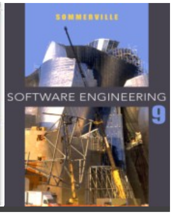




- The **complete specification** is hard to foresee.
  - This becomes problematic when complete specification is required in contract negotiation.
- System **structure tends to degrade** as new increments are added.
  - Unless time and money is spent on extensive **refactoring**, regular changes tend to **corrupt system structure** and increase the cost of incorporating further changes.
- It is hard to identify and effectively design basic **facilities shared** by different parts of the system.
- The process is not visible, **progress is hard to trace**.

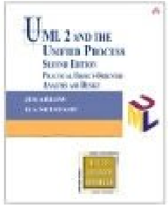
Agile methods

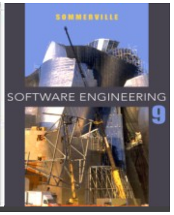




- Agile methods:
  - Focus on the **code** rather than the design
  - Are based on an **iterative approach** to software development
  - Are intended to deliver working software quickly and evolve this quickly to **meet changing requirements**.
- The aim of agile methods is to **reduce overheads in the software process** (e.g. by limiting documentation) and to be able to **respond quickly to changing requirements** without excessive rework.

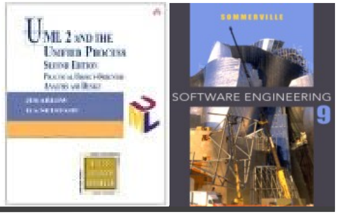
The principles of agile methods





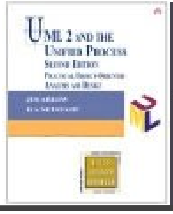
Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

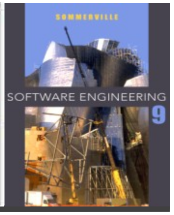
Problems with agile methods



- ✧ It can be difficult to **keep the interest of customers** who are involved in the process.
- ✧ Because of their focus on small, tightly-integrated teams, one needs to be careful when **scaling agile methods** to large systems.
- ✧ **Prioritizing changes** can be difficult where there are **multiple stakeholders**.
- ✧ Maintaining **simplicity** requires extra work.
- ✧ **Contracts** may be a problem as with other approaches to iterative development.

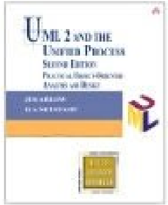
Extreme programming





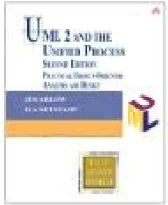
- ✧ Perhaps the best-known and most widely used agile method.
- ✧ Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

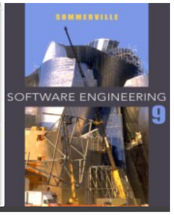
XP and agile principles



- Incremental development is supported through small, regular, **frequent system releases**.
- Customer involvement means **full-time customer engagement** with the team.
- People not process through **pair programming**, **collective ownership** and a process that avoids long working hours.
- Maintaining simplicity through **constant refactoring** of code.

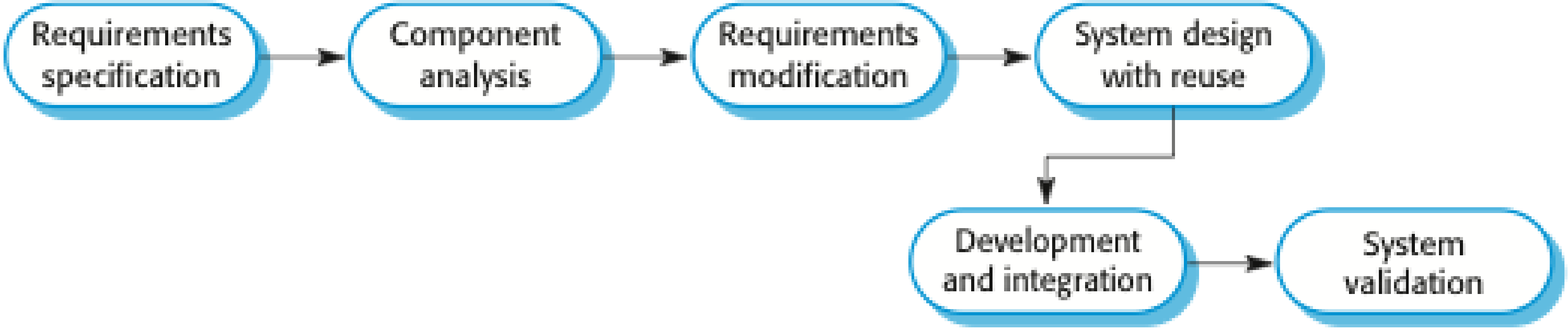
Reuse-oriented software engineering



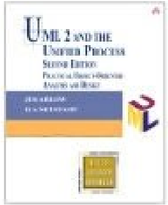


- Based on **systematic reuse** where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.
- Reuse is now the standard approach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


UML 2 AND THE UNIFIED PROCESS

SECOND EDITION

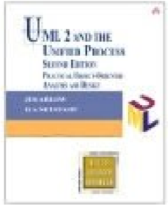
SOMMERVILLE

SOFTWARE ENGINEERING

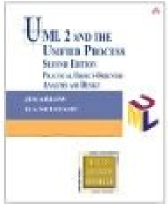
9

- There are many different types of system and each requires appropriate software engineering tools and techniques for their development.
- Software engineering is an engineering discipline that is concerned with all aspects of software production.
  - The high-level activities of specification, analysis and design, implementation, validation and evolution are part of all software processes.
- General process models describe the organization of software processes.
  - Examples of general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.

Key points



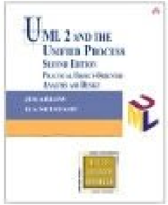
- Processes should include activities to cope with change.
  - This may involve **prototyping** and **incremental delivery**, which help to avoid poor early decisions on requirements and design.
- The Rational Unified Process is a modern generic process model that is
  - **organized into phases** (inception, elaboration, construction and transition)
  - **but separates activities** (requirements, analysis and design, etc.) from these phases.
- Agile methods are incremental development methods that focus on frequent releases, reducing process overheads and emphasize customer involvement.



UML in Software Development

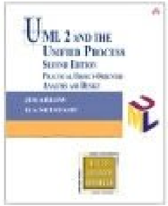
Lecture 1/Part 3

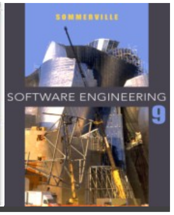
Outline



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

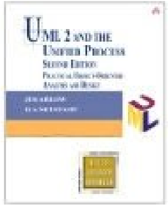
System modeling





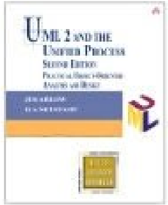
- System modeling is the process of developing **abstract models of a system**, with each model presenting a different view or **perspective** of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations 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**.

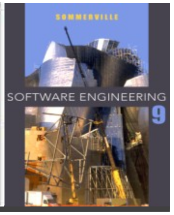
System perspectives



- 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.

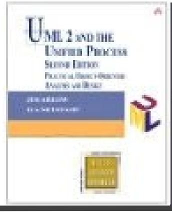
UML diagram types

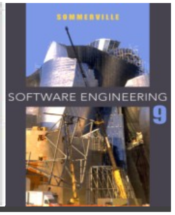




- External perspective
  - **Use case diagram**
- Structural perspective
  - **Class diagram**, Object diagram, Component diagram, Package diagram, Deployment diagram, Composite structure diagram
- Interaction perspective
  - **Sequence diagram**, Communication diagram, Interaction overview diagram,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

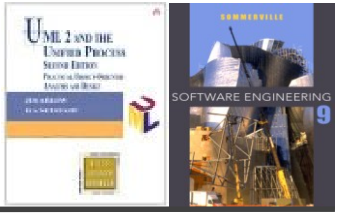
Popular UML diagrams

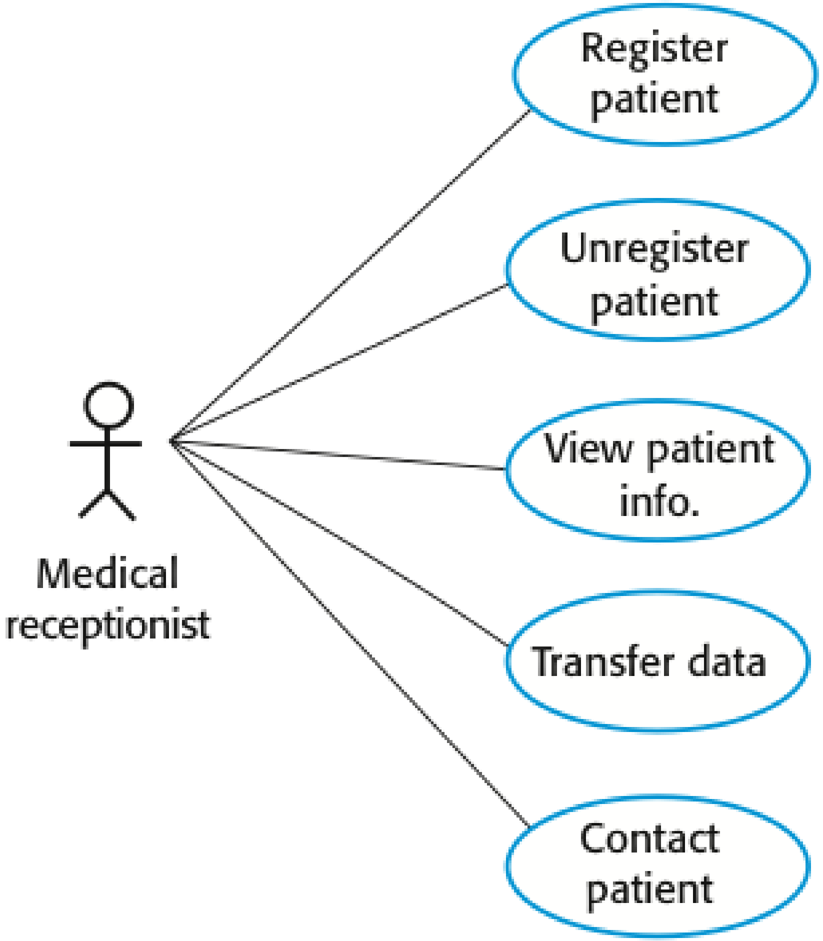




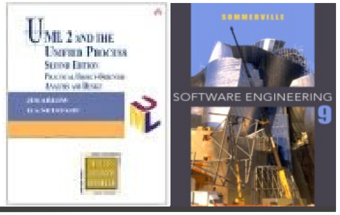
- ✧ **Use case diagrams**, which show the interactions between a system and its environment.
- ✧ **Class diagrams**, which show the object classes in the system and the associations between these classes.
- ✧ **Sequence diagrams**, which show interactions between actors and the system and between system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

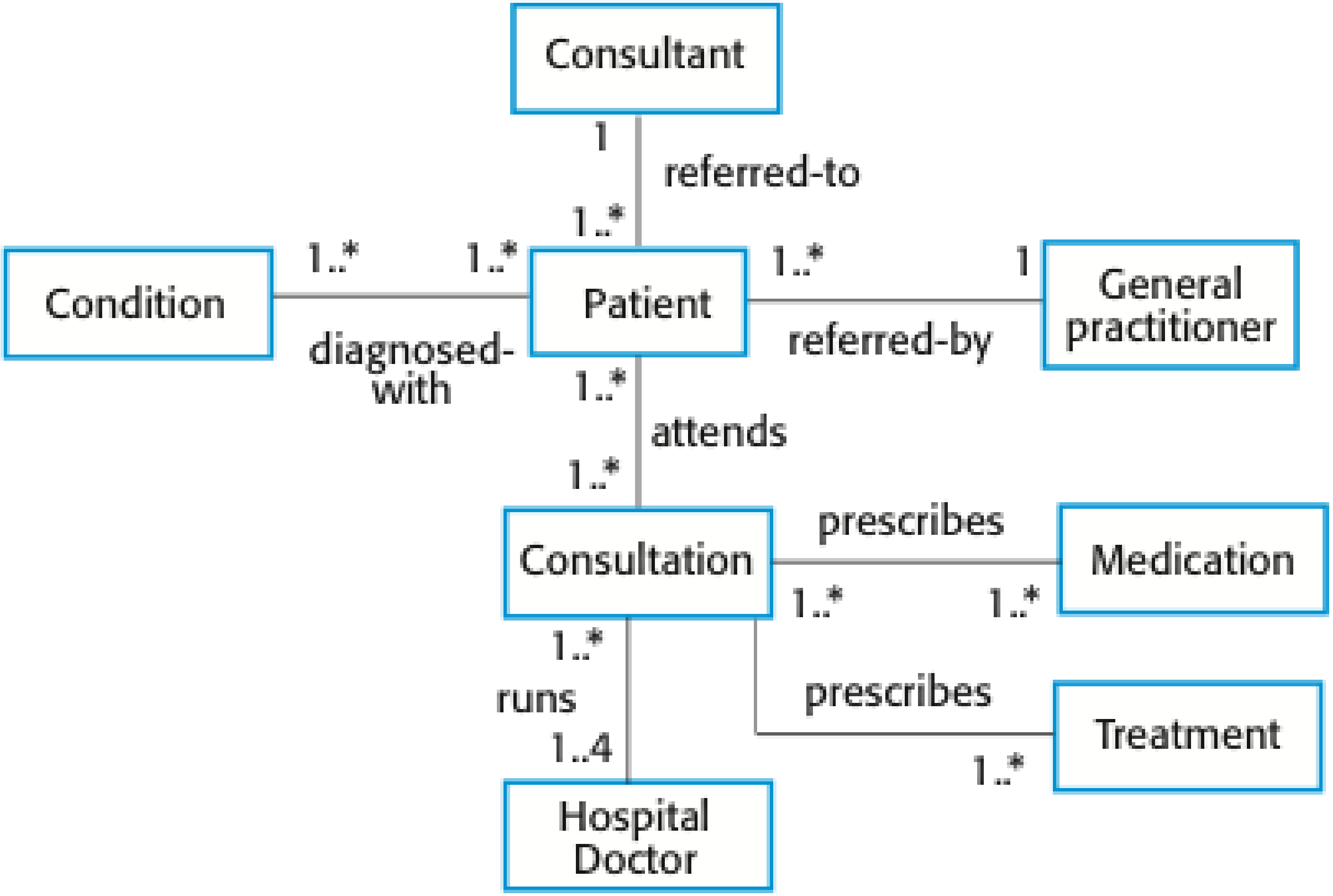
UML Use case diagram: Medical receptionist in health care system



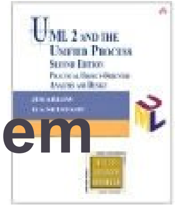


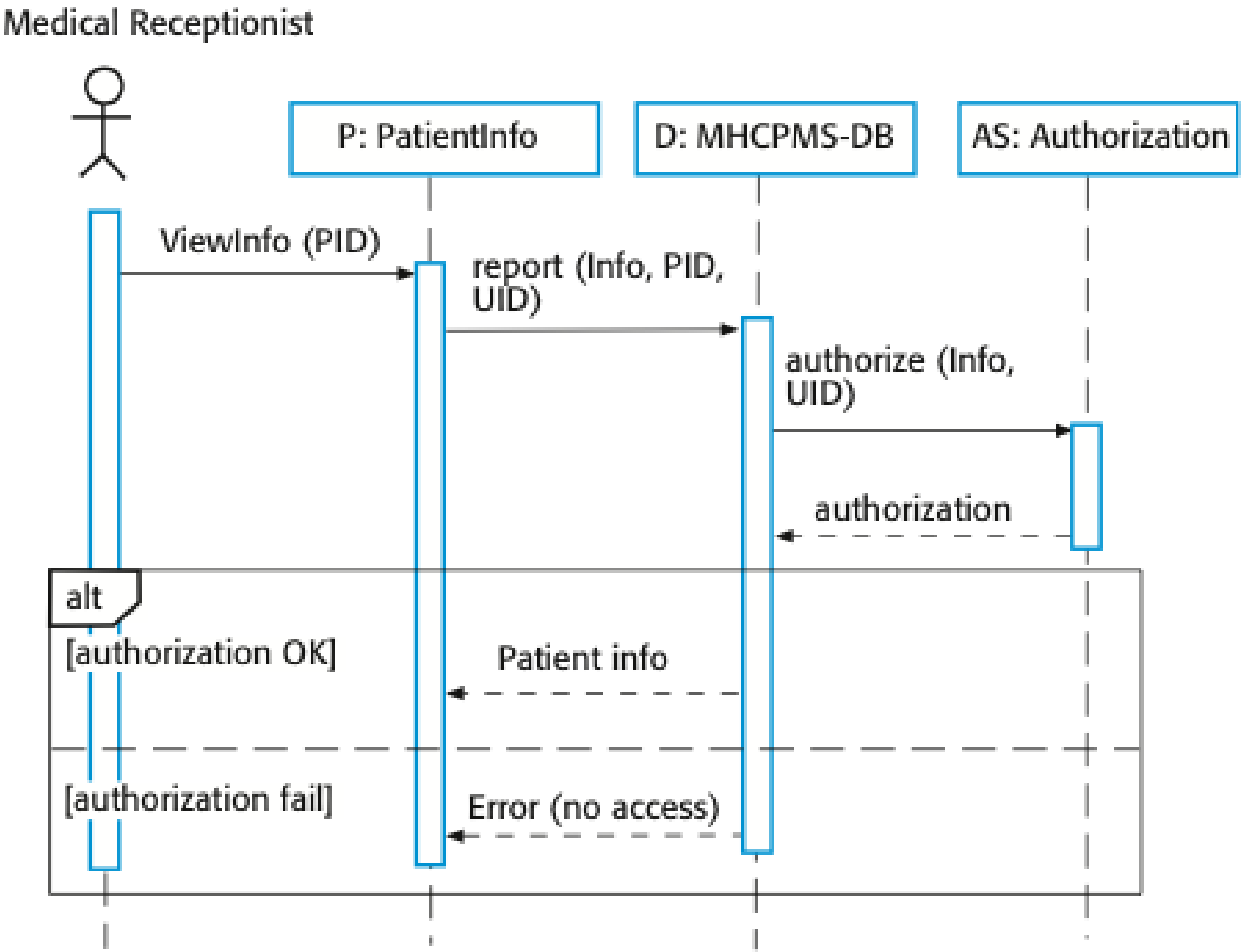
UML Class diagram: Health care system



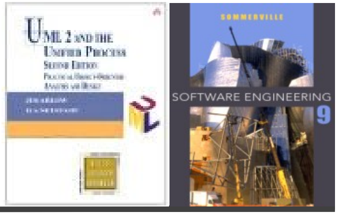


UML Sequence diagram: View patient information in health care system

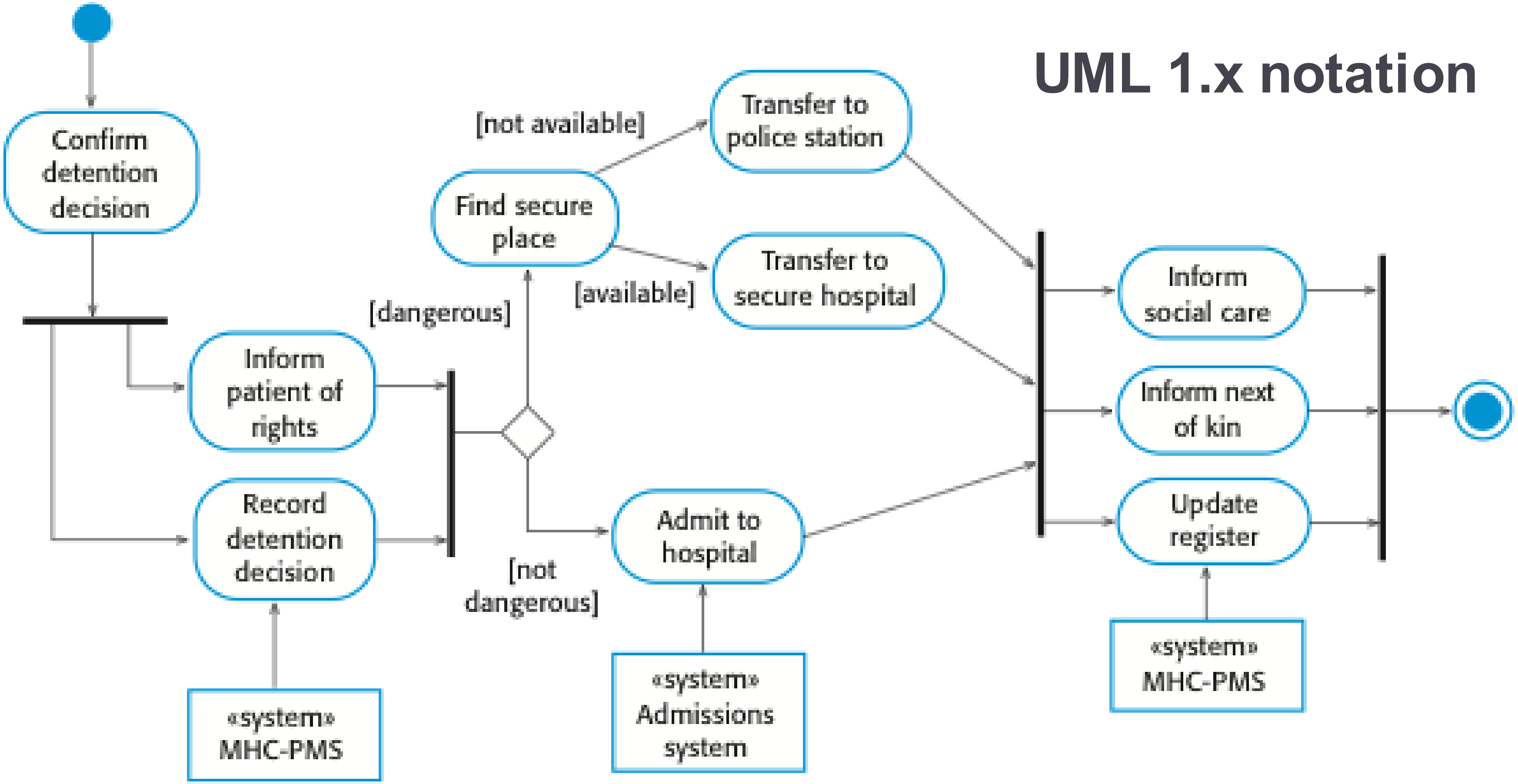




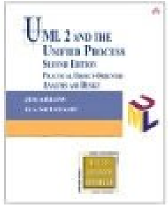
UML Activity diagram:
Process model of involuntary detention



UML 1.x notation



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.